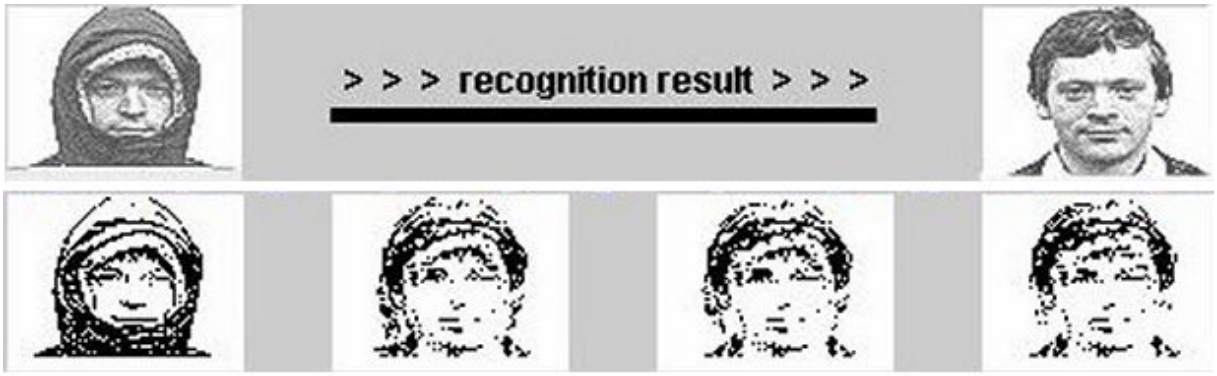
Fachhochschule Hannover
Fachbereich Informatik

Aufbau eines mobilen Systems zur Erkennung von Gesichtern durch die optimale Kombination neuronaler Netze mit Eigenwert- und Matching-Methoden



Forschungsbericht

Prof. Dr. Werner M. Lechner
werner.lechner@inform.fh-hannover.de

März 2005

2

.

Vorwort

Bei der computergestützten Erkennung von Gesichtern kommt es darauf an, Personen, die sich in kontrollierten Bereichen aufhalten bzw. zu diesen Bereichen Zugang erhalten wollen, zuverlässig und schnell zu identifizieren. Durch die zunehmende Bedrohung der öffentlichen Sicherheit hat die Nachfrage nach einer automatischen Personenerkennung mittels Überwachungskameras stark zugenommen. Entsprechende Systeme sind bereits auf einzelnen amerikanischen Flughäfen (Oakland, Boston, Rhode Island), in Spielkasinos zur Identifikation von Falschspielern und nicht zuletzt auch im Zoo in Hannover mit sehr unterschiedlichem Erfolg im Einsatz. Der besondere Vorteil der Gesichtserkennung gegenüber Ausweiskontrollen oder sonstigen biometrischen Verfahren besteht darin, dass die kontrollierten Personen selbst keine Aktionen ausführen müssen und sich unbemerkt im Vorbeigehen überprüfen lassen. Zuverlässigkeit bedeutet in diesem Zusammenhang, dass

- einer berechtigten Person der Zugang in einen geschützten Bereich ermöglicht und
- einer nicht berechtigten Person der Zugang verweigert wird.

Die besondere Problematik besteht darin, dass anscheinend hohe Erkennungsraten, wie z.B. 90%, die dann zu Fehlalarm-Raten von 10% führen, bereits das Abschalten des Erkennungssystems durch das Wachpersonal provozieren. Bei Bankautomaten mit integrierter Gesichtserkennung könnte ein ungünstig gescannter Kontoinhaber dann auf sein eigenes Konto nicht mehr zugreifen, während ein Betrüger mit einem guten Foto des Kontoinhabers das System leicht überwindet.

Die computergestützte Erkennung von Gegenständen oder Lebewesen basiert auf der quantitativen Feststellung einer möglichst weitgehenden Übereinstimmung zwischen einer gespeicherten Graphik und dem momentan erfassten Kamerabild. Im Fall einer einfachen Anwendung gilt eine Person als erkannt, wenn eine gespeicherte Graphik dieser Person existiert, die eine hohe Übereinstimmung mit dem Kamerabild aufweist, d.h. deren Quadratsumme aus den Differenzen der Pixelwerte ein absolutes Minimum annimmt und deren Abstand zum nächst größeren Fehlervektor ausreichend groß ist. Der direkte Bildvergleich setzt jedoch voraus, dass die zu vergleichenden Graphiken identische Aufnahmewinkel, Skalierungen, Beleuchtungsbedingungen, usw. aufweisen und vor dem gleichen Hintergrund aufgenommen werden. Doch als eigentliche Schwierigkeit kommt hinzu, dass die momentanen Kamera-Scans ein und derselben Person sehr unterschiedlich ausfallen können, denn die entsprechende Person kann ihr äußeres Erscheinungsbild im Vergleich zur gespeicherten Kameraaufnahme beliebig verändern.

Der mathematische Aufwand für die Erkennungsalgorithmen ist begrenzt, denn die eigentliche Erkennung soll innerhalb von wenigen Sekunden abgeschlossen sein, d.h. die Algorithmen sollten Echtzeitfähigkeit aufweisen. Zusammen fassend lassen sich somit drei Kriterien aufstellen, denen ein Gesichtserkennungssystem genügen muss:

Kriterium 1	$\Rightarrow$	Zuverlässigkeitsrate deutlich über 90%
Kriterium 2	$\Rightarrow$	Robustheit gegenüber Täuschungsversuchen
Kriterium 3	$\Rightarrow$	Echtzeitfähige Algorithmen

Tab.: 1: Kriterien eines Gesichtserkennungssystems

Mit Blick auf diese Problematiken eignen sich zur Erkennung hauptsächlich jene mathematischen Verfahren, die möglichst invariant gegenüber den aufgelisteten Kriterien sind.

Zur Erkennung von Gesichtern lassen sich z.B. mittels der Graphentheorie die Abstände von Augen, Nase und Mundwinkel ermitteln und als Basis des Bildvergleichs heranziehen. Oder man wandelt die Pixelbilder mittels geeigneter Transformationen in eine komprimierte Darstellung um, die weitgehend invariant gegenüber geometrischen Variationen ist. Außerdem ist es hilfreich, bei der Gesichtserkennung gleich mehrere Aufnahmen mit unterschiedlichen Aufnahmewinkeln bzw. Entfernungen abzuspeichern. Die Literatur zur Bilderkennung beschreibt eine Vielzahl von algorithmischen Verfahren. Die bekanntesten Verfahren sind:

- Vergleich von signifikanten Bildausschnitten (templates)
- Vergleich geometrischer Merkmale eines Gesichtes, wie Abstände und Winkel
- Vergleich auf der Basis 2-dimensionaler Integraltransformationen des Bildmaterials
- Neuronale Netze mit den unterschiedlichsten Topologien
- Parameter-Transformationen, wie z.B. die allgemeine Hough-Transformation
- Vergleich von elastischen Graphen mittels Gabor-Wavelets
- Vergleich von „Eigengesichtern“ gemäß der Eigenwertmethode der linearen Algebra

Im organisatorischen Rahmen einer zweisemestrigen studentischen Projektarbeit führte eine Gruppe aus 12 Studierenden des Fachbereichs Informatik der Fachhochschule Hannover unter der Leitung des Autors eine zweisemestrige Projektarbeit durch und untersuchte dabei die Eigenschaften der bekanntesten Algorithmen. Für den Testbetrieb beschaffte der Fachbereich Informatik eine leistungsfähige und in der Bewegung steuerbare Canon-Kamera, ein „WiFiBot“-Fahrzeug mit feststehender Kamera und WLAN-Link zur Steuerung und Bildübertragung sowie zwei Computer mit Dual-Prozessor Boards. Im Rahmen des genehmigten HBFG-Antrages „Verteilte Systeme“ ist die Erweiterung der beiden Rechner auf ein PC-Cluster erfolgt. Damit kann der enorme Rechenaufwand bewältigt werden, der bei der Entwicklung der umfangreichen Software mit studentischen Arbeitsgruppen anfällt. Für das offline-Training der neuronalen Netze und grundlegende Untersuchungen steht ein mit 16 Prozessoren bestückter HPCLine Siemens Parallelrechner (Abb. 1) des Fachbereichs Informatik der Fachhochschule Hannover zur Verfügung.

Abb.: 1: Hardware für das Forschungsprojekt

Der Autor befasst sich seit vielen Jahren mit der Anwendung neuronaler Netze zur Lösung von technischen Problemen und gibt in den Lehrveranstaltungen „Künstliche Intelligenz“, „Softcomputing“ und „Parallelverarbeitung“ sowie bei Projekt- , Semester- und Diplomarbeiten dieses Wissen an die Studierenden im Fachbereich Informatik der Fachhochschule Hannover weiter. In der Publikation

> „In-Flight Wind Prediction Algorithm using Recurrent Neural Networks,
> Neural Networks in Engineering Systems“,
> Proceedings of the International Conference EANN 97,
> Stockholm Schweden,1997

ging es darum, den Treibstoffverbrauch von Transportflugzeugen zu minimieren und die geplanten Flugzeiten exakt einzuhalten. Mit den wissenschaftlichen Angestellten des Fachbereichs Informatik besteht eine enge Zusammenarbeit. Die neueste Publikation des Authors, die er zusammen mit Herrn Dipl.-Ing. Frank Müller erstellt hat, trägt den Titel:

> „Intelligent Control of Autonomous Six-Legged Robots by Neural Networks“,
> International Federation of Automatic Control,
> IFAC International Conference on Intelligent Control Systems and Signal Processing, ICONS 2003, April 2003, University of Algarve

Diese Publikation demonstriert das Potenzial einer künstlichen Intelligenz am Beispiel der Steuerung autonomer sechsbeiniger Kleinroboter, die Hindernissen ausweichen und gegenüber identisch programmierten Kleinrobotern sinnvolles, kollisionsfreies Verhalten aufzeigen, obwohl insgesamt nur Grundmuster für einfachste Bewegungsabläufe trainiert wurden. Im Review für die Vortragsanmeldung schreibt einer der Gutachter zu der eingereichten ICAS-Veröffentlichung aus dem Bereich der neuronalen Netze (NN):

> „This is a very good contribution. It demonstrates NN ability to learn and to generalise the learned knowledge hence being an intelligent object. I have seen very few papers being so transparent in illustrating intelligence of NN.“

Der Author erhielt im Wintersemester 2002/2003 von der Forschungskommission der Fachhochschule Hannover die Gelegenheit zu einem Forschungssemester und baute ein mobiles System zur Gesichtserkennung auf der Basis rückgekoppelter neuronaler Netzwerke auf. Das System ist gekennzeichnet durch eine optimale Kombination unterschiedlicher Methoden der Bilderkennung, die sich gegenseitig in ihren günstigen Eigenschaften ergänzen. Dadurch entstand ein leistungsfähiges System, das den Kriterien in Tab. 1 genügt. Der vorliegende Bericht dokumentiert die Ergebnisse dieser Forschungsarbeiten.

Hannover, im Februar 2005

Prof. Dr. Werner Lechner

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Die Problematik der Gesichtserkennung

Die besondere Problematik einer Personenerkennung besteht darin, dass die Kamerabilder ein und derselben Person sehr unterschiedlich ausfallen können und trotzdem eine zuverlässige Erkennung gewährleistet sein muss. Das menschliche Gehirn löst dieses Problem durch den Vergleich der Abstände einzelner Gesichtsmerkmale und durch die Einbeziehung weiterer biometrischer Merkmale, wie z.B. der Körpergröße, des Alters und ähnlicher Parameter. Ungeklärt ist allerdings bis heute, ob der Mensch eine Person als Ganzes speichert (holistisch) oder lediglich nur als eine Reihe von konfiguralen Parametern, wie z.B. den geometrischen Abständen zwischen Augen, Nase oder Mund und den Werten für die allgemeinen Körperproportionen. Beeindruckend ist in jedem Fall die kurze Rechenzeit von etwa 100ms, die unser Gehirn für eine Personenerkennung benötigt.

Will man die Vorteile der Personenerkennung technisch nutzen und dem gesteigerten Sicherheitsbedürfnis der Gesellschaft Rechnung tragen, dann muss es darum gehen, die Zuverlässigkeit und die Robustheit der vorhandenen Systeme deutlich zu erhöhen. Unter sicherheitskritischen Aspekten reichen selbst hohe Erkennungsraten von z.B. 90% nicht aus und wären als Systemversagen zu interpretieren. Die zu geringe Zuverlässigkeit dieser Systeme hat die anfängliche Begeisterung für die Gesichtserkennung stark gedämpft und große Kreditinstitute, wie z.B. die Sparkassen-Finanzgruppe, die sich von der Gesichts- und Unterschriftenerkennung eine Ablösung der aufwändigen PIN/TAN Nummernverwaltung erhofften, stellten frustriert die finanzielle Unterstützung im Großprojekt BioTrust ein.

Von besonderer Bedeutung im Zusammenhang mit der Personenerkennung sind mathematische Verfahren, die eine möglichst große Invarianz gegenüber den Momenten Drehung, Verschiebung und Größenänderung des Kamerabildes auszeichnet. Mit Blick auf die angestrebte hohe Zuverlässigkeit einer Gesichtserkennung sollten sich die Parameter Bildhelligkeit, Bildkontrast, Bildschärfe, Gesichtsausdruck, Hintergrund, Kopfposition, Schattenbildung und Verdeckungen im Gesicht nur in vernachlässigbarer Weise auf der Ebene des Bildvergleichs auswirken.

Neben der Problematik einer hohen Erkennungsrate, besteht die zusätzliche Forderung nach der Robustheit gegenüber Täuschungsversuchen. Ein praxistaugliches Erkennungssystem darf sich nicht durch hochwertige Fotos einer authorisierten Person täuschen lassen. In einer Testreihe, welche die bekannte Computerzeitschrift „ct“ im Frühjahr 2003 durchführte, versagten alle getesteten preiswerten biometrischen Erkennungssysteme. Tatsächlich stellt nämlich die Erkennung von Gesichtern eine anspruchsvolle Aufgabe dar, die in der Regel auch einen enormen Rechenaufwand erfordert und mit low-cost Systemen nicht zu realisieren ist.

1.2 Die Evolution des biologischen Sehens

Das menschliche Auge erfasst die Welt mit ca. 126 Millionen Rezeptoren. Man unterscheidet zwischen den ca. 120 Millionen Stäbchen für das Dämmerungssehen und den ca. 6 Millionen Zäpfchen in der Netzhautmitte für das Farbsehen in den Farben blau, grün und rot. Begonnen hat diese Entwicklung jedoch vor Millionen von Jahren mit Einzellern, für die es nur hell oder dunkel gab. Ein Regenwurm sieht ebenfalls nur hell oder dunkel und entscheidet sich dann für eine Bewegung in Richtung der dunklen Welt. Schlangen besitzen ein Linsenauge, wobei die Netzhautzellen nur auf Änderungen der Lichtstärke reagieren. Deshalb muss die Schlange ruhig verharren, damit die Rezeptoren den bewegten Schattenwurf registieren, den ein Beutetier auf der Netzhaut der Schlange erzeugt. Aus der Größe des betroffenen Netzhautgebietes und der Linsenfokussierung, welche der Entfernung zum Beutetier entspricht, berechnet das Gehirn der Schlange die Größe des Beutetieres. Pflanzenfresser brauchen zusätzliche Farbrezeptoren zur Erfassung der Farbe grün.

Die Evolutionsgeschichte des Sehens zielt auf ein Erkennen der Umwelt, deren Bilder auf der Netzhaut entstehen. Es geht also nicht darum, ein realitätsnahes Foto an die Neuronen des Gehirns zu liefern, sondern das Sehen dient einzig der Fortpflanzung und dem Überlebenskampf:

Lichtempfindlichkeit	⇒	Unterscheidung von Himmel oder Erde
Bewegungssehen	⇒	Unterscheidung von lebendig oder tot
Farbsehen	⇒	Unterscheidung von Pflanze oder Tier
Größensehen	⇒	Unterscheidung von Jäger oder Beute

Tab.: 1.1: Evolution des Sehens (nach [10])

Beim Menschen ist dieser Differenzierungsprozess sehr weit fortgeschritten. Wir können die Empfindlichkeit der Rezeptoren an Helligkeits- und Kontrastschwankungen anpassen. Unser Gehirn extrahiert aus den Einzelbildern beider Augen eine räumliche Information. Außerdem zittern unsere Augen mit der Amplitude des Rezeptorabstandes ständig mit ca. 100 Hz, um uns ein Sehen ruhender Objekte zu ermöglichen. Mit Medikamenten (Novokain) läßt sich dieses Zittern abstellen und dann kann man als Mensch erleben, wie eine Schlange sieht: In einem hellen Nebel tauchen Objekte auf und verschwinden wieder, ohne dass ein Zusammenhang erkennbar wäre. Nur was sich bewegt, nimmt vor dem gelähmten Auge eine Gestalt an.

Durch die Herausforderungen des Alltags verbessert der Mensch ständig seine Fähigkeiten zur Unterscheidung von Objekten. Ein Landwirt, der jeden Tag mit seinen Kühen beschäftigt ist, kann alle Tiere unterscheiden. Wer Zwillinge häufig zusammen sieht, kann bald beide Personen problemlos auseinander halten. Und wer behauptet, Chinesen sähen alle gleich aus, der hat bisher keine Gelegenheit erhalten, diese Unterscheidungsfähigkeit zu entwickeln, denn tatsächlich sind chinesische Gesichtszüge objektiv betrachtet genau so unterschiedlich wie europäische.

Zur Wiedererkennung von Objekten muss ein Lebewesen in der Lage sein, die erkannten Merkmale zu speichern. Ein Mensch erkennt eine Person wieder, wenn er im Bild, das auf der Netzhaut entsteht, bekannte Merkmale wahrnimmt. Im Traum aktiviert der Mensch diese Merkmale und es entstehen wieder Bilder, die sich jedoch von den tatsächlichen Bildern deutlich unterscheiden können. Mathematisch betrachtet handelt es sich dabei um eine mehrdeutige Hin- und Rücktransformation auf die Ebene der Merkmale. Dies

erklärt auch, weshalb es einem Menschen außerordentlich schwer fällt, eine Zeichnung dessen anzufertigen, was er wirklich gesehen hat.

Das menschliche Sehvermögen ist eben kein Fotoapparat, sondern dient vor dem Hintergrund der Evolution dem Erkennen von Objekten, vor allem jenen, die für ein Überleben wichtig sind oder waren. Diese Tatsache lässt sich am Beispiel von Vexierbilder veranschaulichen. Bei der Betrachtung der Abb. 1.1 kann man im linken Bild eine Vase oder zwei Gesichter erkennen. Das mittlere und rechte Bild in der Abb. 1.1 stellt anscheinend unterschiedliche Gesichter dar, obwohl es sich um identische, jedoch gedrehte Skizzen handelt.

Abb.: 1.1: Vexierbilder

1.3 Grundlegende Definitionen

Für das grundlegende Verständnis und die quantitative Bewertung der Methoden zur Personenerkennung sind eine Reihe von speziellen Begriffen hilfreich. Grundsätzlich unterscheidet man zwischen zwei Fehlerarten:

1. Eine Person erhält Zutritt in einen geschützten Bereich, obwohl diese Person dazu nicht berechtigt ist.

2. Einer Person wird der Zutritt verweigert, obwohl diese Person berechtigt ist, den geschützten Bereich zu betreten.

Zwischen den beiden grundsätzlichen Fehlerarten besteht mit Blick auf den praktischen Einsatz derartiger Systeme ein großer Unterschied. Bei der ersten Fehlerart erhält eine nichtberechtigte Person Zutritt, was großen Schaden verursachen kann, wie z.B. im Falle eines Bankautomaten. Der Sinn des Gesichtserkennungssystems sollte doch gerade darin bestehen, genau diesen Fall zu verhindern.

Die zweite Fehlerursache löst nicht nur Fehlalarme aus, was für alle Beteiligten schon belastend genug ist, sondern die berechtigte Person erhält keinen Zutritt. Anschließend ist eine persönliche Identitätsprüfung vorzunehmen. Eine Person, der wegen einer zu geringen Erkennungsrate der berechtigte direkte Zutritt schon mehrfach verweigert wurde, drängt vermutlich auf das Abschalten des Überwachungssystems und das Dienstpersonal neigt dann dazu, das ganze Überwachungssystem nicht mehr ernst zu nehmen. Eine exakte Definition der Erkennungsraten ist daher von größter Bedeutung.

Die folgenden Tabellen **??** und **??** beschreiben diese Fehlerquellen einer Personenerkennung durch eindeutige, quantitative Definitionen. Damit lassen sich dann Erkennungssysteme bewerten und in ihrer Leistungsfähigkeit quantitativ messen. Obwohl in den Tabellen nur von Personenerkennung die Rede ist, gelten die Definitionen für beliebige zu erkennende Objekte.

Detektion	Personendetektion meint das Feststellen, ob sich in einem Kamerabild überhaupt eine Person befindet. Das ist im Einzelfall keine triviale Aufgabe.
Lokalisierung	Mit der Lokalisierung grenzt man das Kamerabild auf einen Bildausschnitt ein, in dem sich die zu erkennende Person befindet.
Normalisierung	Das Bild der Person wird auf Standardgröße skaliert, auf senkrechte Orientierung ausgerichtet und zentriert.
Tracking	Unter Tracking versteht man das Verfolgen einer Person im laufenden Kamerabild. Dies gelingt durch die Kombination von Personendetektion und Personenlokalisierung.
Authentifikation	Allgemeiner Ausdruck für die Bestätigung der Identität einer Person.
Verifikation	Verifikation meint die Bestätigung der angegebenen Identität, d.h. das Kamerabild wird mit nur einem der gespeicherten Bilder verglichen, um festzustellen, ob es sich tatsächlich um diejenige Person handelt, für die sie sich ausgibt.
Identifikation	Unter Identifikation versteht man das Feststellen der Identität einer Person, d.h. das Kamerabild wird mit allen gespeicherten Bildern verglichen um festzustellen, um welche Person es sich handelt.

Tab.: 1.2: Definitionen A

Number of False Acceptances (NFA)	Der Wert von NFA erfasst die Anzahl der fälschlicherweise für authorisiert erkannten Personen.
Number of Imposter Attemps (NIA)	Der Wert von NIA beschreibt die Anzahl der Zutrittsversuche nicht authorisierter Personen.
False Acception Rate (FAR) $FAR = \frac{NFA}{NIA} \cdot 100\%$	Unter der FAR versteht man die fälschlicherweise als berechtigt erkannten, tatsächlich jedoch nicht authorisierten Personen.
Number of False Rejections (NFR)	NFR beschreibt die Anzahl der fälschlicherweise zurückgewiesenen Personen, also jener Personen, die eigentlich authorisiert sind.
Number of Enroll Attemps (NEA)	Beim NEA Wert handelt es sich um die Anzahl der Zutrittsversuche der berechtigten Personen.
False Rejection Rate (FRR) $FRR = \frac{NFR}{NEA} \cdot 100\%$	Die FRR quantifiziert den relativen prozentualen Anteil der authorisierten, jedoch fälschlicherweise abgewiesenen Personen

Tab.: 1.3: Definitionen B

Beispiel 1:

Eine Anzahl von 1000 Personen versucht an einem Geldautomaten mit Personenerkennung Geld abzuheben. Unter den 1000 Personen befinden sich 100 nicht berechtigte Personen. Das Erkennungssystem arbeitet mit jeweils 10% FAR- bzw. FRR-Fehlerrate. Die Abb. 1.2 stellt die Zutrittsversuche der 900 berechtigten Personen dar. Auf Grund der FRR von 10% können 90 berechtigte Personen nicht auf ihr Konto zugreifen.

Untolerierbar ist das in der Abb. 1.3 erkennbare Ergebnis: Wegen der FAR von 10% gelingt es 10 Personen unberechtigt Geld abzuheben. Da tröstet es wenig, wenn andererseits 90 von 100 Betrügern der Zugang verwehrt wird.

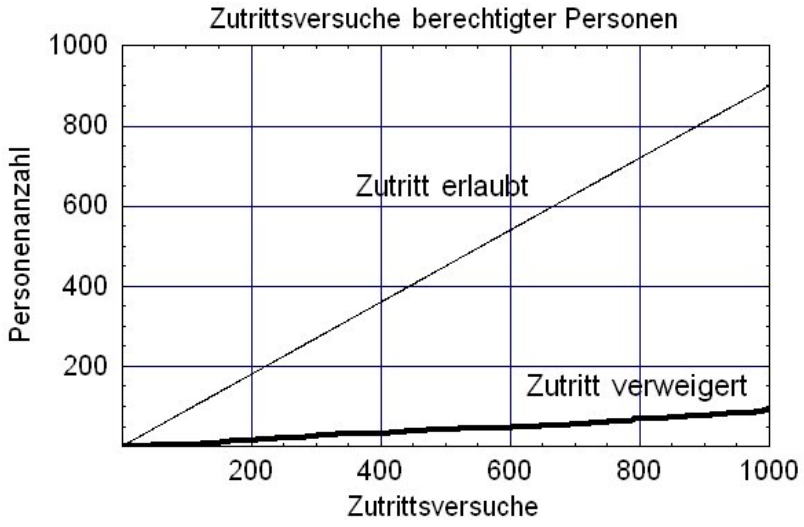


Abb.: 1.2: Graphische Darstellung der Ergebnisse für berechtigte Personen

Welche FAR bzw. FRR akzeptiert werden können, hängt von der konkreten Aufgabenstellung ab. Im Bereich der Geldautomaten ist eine FAR von 0% erforderlich. Um Ärger mit unzufriedenen Kunden zu vermeiden, sollte auch die FAR deutlich unterhalb einem Prozent liegen.

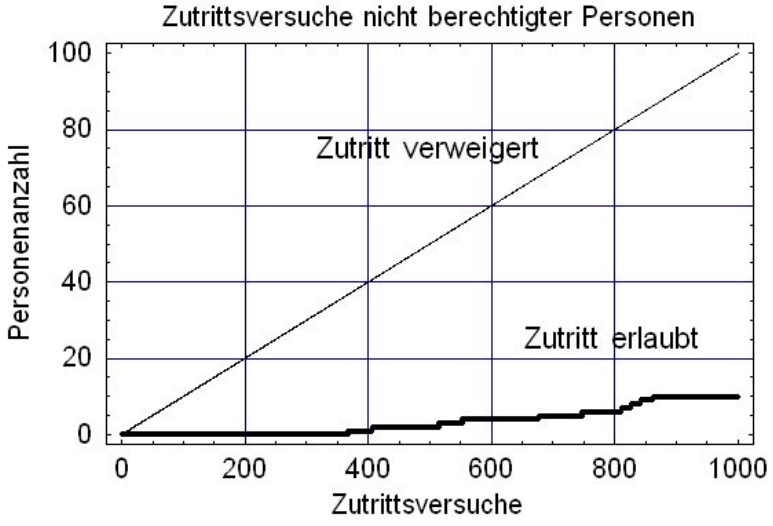


Abb.: 1.3: Graphische Darstellung der Ergebnisse für nicht berechtigte Personen

1.4 Templates und Gesichtsparameter

Bei den Verfahren der Gesichtserkennung spielen Templates und Gesichtsparameter eine wichtige Rolle, denn häufig ist es günstiger nicht das Bild als Ganzes zu vergleichen, sondern einen Einzelvergleich markanter Bildausschnitte, der Templates, durchzuführen.

Ein Template sollte einen möglichst hohen Informationsgehalt besitzen. Daher eignen sich z.B. bei der Auswertung von Gesichtsbildern die Augen-, Nasen- und Mundpartie. Für jedes dieser Templates lassen sich Parameter bestimmen und aus den Abweichungen zum abgespeicherten Parameter des Original-Teilbildes eine quadratische Fehlersumme ermitteln. Ein Gesicht gilt als erkannt, wenn die Fehlersumme über alle Templates ein Minimum im Vergleich zu allen anderen gespeicherten Bildern annimmt.

Problematisch gestaltet sich bei dieser Methode die Normierung der Gesichtsaufnahme, denn bei der Aufnahme eines gedrehten oder geneigten Gesichtes verändern sich viele der geometrischen Daten. Erschwerend kommt hinzu, dass die markanten Teile, d.h. also jene Bildausschnitte, die einen hohen Informationsgehalt aufweisen, überhaupt erst detektiert und lokalisiert werden müssen ([7]). Die Abb. 1.4 stellt das berühmte Bild der Mona Lisa dar und zeigt Beispiele möglicher Templates.

Der mathematisch einfache und direkte Bildvergleich reagiert sehr empfindlich auf Helligkeits- und Kontraständerungen und eine präzise und praktisch kaum realisierbare Normierung der Bildscans wäre Grundvoraussetzung. Selbstverständlich müssen sich auch die Inhalte der zu vergleichenden Bilder an der jeweils gleichen Position befinden.

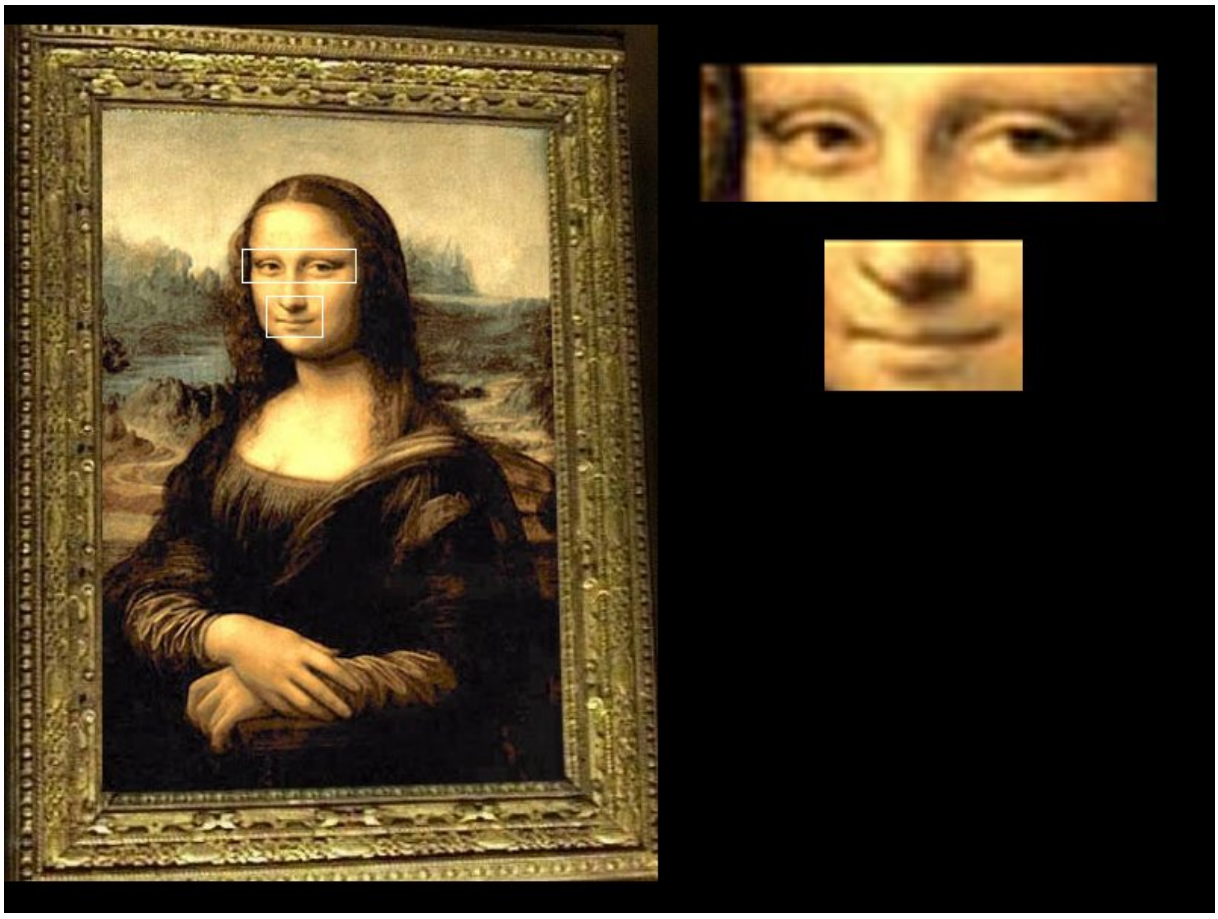


Abb.: 1.4: Mona Lisa

Als anschauliches Beispiel stellt die Abb. 1.5 ein gespeichertes und ein gescanntes graphisches Objekt dar. Das menschliche Auge bzw. das Bildverarbeitungssystem des Gehirns erkennt klar, dass es sich bei den Graphiken der Abb. 1.5 um ein identisches Objekt handelt. Der numerische Pixelvergleich würde dagegen nirgends eine Übereinstimmung zeigen.

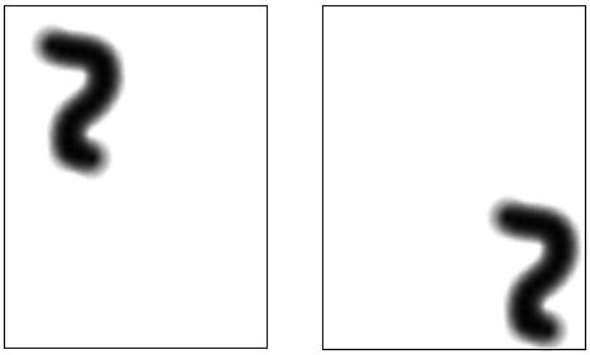


Abb.: 1.5: Gespeichertes und gescanntes Bild

Als Lösungsweg bietet sich nun an, die gescannte Graphik, die in der Regel die schlechtere Normierung aufweist, so lange zu filtern, zu verschieben, zu drehen und zu zoomen, bis der Pixelvergleich eine maximale Übereinstimmung aufweist. Das hört sich leicht an, ist jedoch äußert schwierig, denn die beiden Lösungsschritte,

1. das Herstellen einer vergleichsfähigen Struktur der gescannten Graphik (Filterung, Normierung, Transformation) und

2. der Pixelvergleich (min. Fehlerquadrate, Eigenwert-Methode, Neuronale Netze, Parametervergleich)

sind voneinander abhängig. Ohne eine erfolgreich durchgeführte Normierung macht es keinen Sinn, einen aufwändigen und leistungsstarken Pixelvergleich durchzuführen. Die Normierung wiederum ist keine triviale Aufgabe, denn die Werte für die Filterung und die geometrischen Transformationsparameter sind unbekannt und zu ihrer Bestimmung ist wiederum der Pixelvergleich erforderlich. Man bezeichnet die entsprechenden Lösungsverfahren als „non-rigid point matching“ Problem, wobei „rigid“ einen starren Körper bezeichnet. Ein allgemeines und in sich geschlossenes mathematisches Verfahren, das beide Aufgaben löst, ist leider noch nicht erfunden.

Bei der Bestimmung der Knoten können erhebliche Probleme auftreten. Häufig liegt nur ein verformter und verrauschter Bildscan vor, dessen extrahierte Punkte, unter denen sich auch Ausreisser befinden können, nur teilweise mit den gespeicherten Bildpunkten korrespondieren.

Eine zentrale Bedeutung nimmt in diesem Zusammenhang die „Korrespondenz-Matrix“ K ein. Sie beschreibt den zu bestimmenden Zusammenhang zwischen den n Pixelpunkten x der Orginalgraphik und den m Pixelpunkten y des Bildscans. Die in der Gleichung (1.1) dargestellte Korrespondenz-Matrix sagt aus, dass der Punkt x2 in der ersten Graphik dem Punkt y1 in der zweiten Graphik entspricht. Jede Zeile bzw. Spalte enthält nur eine logische Eins, denn jeder Punkt hat immer nur eine Entsprechung in der anderen Graphik. Punkte, die keine Entsprechung in der anderen Graphik haben, interpretiert

man als Ausreisser und legt sie in der äußersten Spalte bzw. in der letzten Zeile als Null ab.

$$K = \begin{pmatrix} xy & y1 & y2 & y3 & \dots & ym & Ausreisser \\ x1 & 0 & 1 & 0 & \dots & 0 & 0 \\ x2 & 1 & 0 & 0 & \dots & 0 & 0 \\ x3 & 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ xn & 0 & 0 & 0 & \dots & 1 & 0 \\ Ausreisser & 0 & 0 & 0 & \dots & 0 & 0 \end{pmatrix} \tag{1.1}$$

Ärgerlich ist freilich, dass diese Korrespondenz-Matrix beim Vergleich zweier Graphiken nicht bekannt ist. Bei einer geringen Anzahl von Pixeln lassen sich alle Kombinationsmöglichkeiten der Korrespondenz-Matrix auswerten, denn die Anzahl der Permutationen p liegt bei p=n!. Gültig ist jeweils jene Korrespondenz-Matrix, welche die geringste Summe der Fehlerquadrate der einzelnen Pixelpunkte bei der momentan vorliegenden Normierung aufweist.

Einen teilweisen Ausweg aus dieser Problematik bietet eine Abstandsberechnung der Punkte x und y nach der Hausdorff-Formel (1.2) [6]:

$$H(X,Y) = max(h(X,Y), h(Y,X)) \tag{1.2}$$
$$h(X,Y) = max_x min_y ||x - y|| \tag{1.3}$$

Die Hausdorff-Formel (1.2) bestimmt zunächst den kürzesten Abstand eines Punktes x_i aus der Punktmenge X zu allen Punkten y der Punktmenge Y und liefert dann den maximalen Wert aller dieser Abstände $h(X,Y)$ zurück. Anschließend wiederholt man die Berechnungen, beginnt nun aber mit der Punktmenge Y. Als Endergebnis folgt dann der größere der beiden Zahlenwerte, der ein Maß für die Übereinstimmung zweier Punktfolgen X, Y darstellt. Auch in diesem Fall darf die Anzahl der Punkte nicht beliebig große Werte annehmen, damit die Berechnung der Hausdorff-Abstände in endlicher Zeit erfolgen kann. Der ganz besondere Vorteil der Hausdorff-Formel besteht darin, dass die unbekannte Reihenfolge der Punkte nicht in die Berechnungen eingeht.

Im Zusammenhang mit der Erkennung von Gesichtern ist es günstig, ein Gesicht durch die geometrischen Beziehungen zwischen markanten Teilen zu beschreiben. Im Hinblick auf die relativen Abstände zwischen Augen, Nase und Mund lassen sich insgesamt mehr als 22 Abstände und Radien aus dem menschlichen Gesicht ableiten. Die entsprechenden Zahlenwerte des Bildscans speichert man in einem Vektor ab, der dann die mathematische Basis für den Bildvergleich darstellt. Ein Gesicht gilt als erkannt, wenn sein Merkmalsvektor den minimalen Abstand zu einem gespeicherten Merkmalsvektor aufweist.

Das Verfahren von Wiskott [34] geht von einem symmetrischen Anfangsgitternetz aus, dessen Knotenpositionen an das Kamerbild in optimaler Weise anzupassen sind. Die Knoten des Graphen liegen über den Augäpfeln, auf der Nasenspitze, an den Mundwinkeln, an der Kinnspitze, usw. Daher bildet der Graph die Merkmalstrukturen eines Gesichtes gut ab. An der Stelle der Knoten erfolgt eine Wavelet-Tranformation auf der Basis eines Gaborkerns.

Das Aufsuchen der Lage z.B. des Augenpaares ist allerdings keine triviale Aufgabe [7]. Ein überlagertes Rauschen darf nicht als Augenpunkt interpretiert werden. Eine Vorfilterung zur Glättung der Pixelwerte reduziert den Informationsgehalt einer Aufnahme und erschwert die nachfolgende Kantenbestimmung.

Die Gesichtserkennung auf der Basis von Gitternetzen erfordert eine sorgfältige Normierung der Gesichtsaufnahmen, denn in einem gedrehten oder geneigten Gesicht verändern sich alle geometrischen Parameter. Erschwerend kommt noch dazu, dass die markanten Teile, d.h. also jene Bildausschnitte, die einen hohen Informationsgehalt aufweisen, detektiert und lokalisiert werden müssen.

Für die unterschiedlichen Gesichtsansichten existieren Grundformen dieses Graphen. Die Abb. 1.6 stellt beispielhaft elementare Graphen für drei Ansichten eines Gesichtes dar.

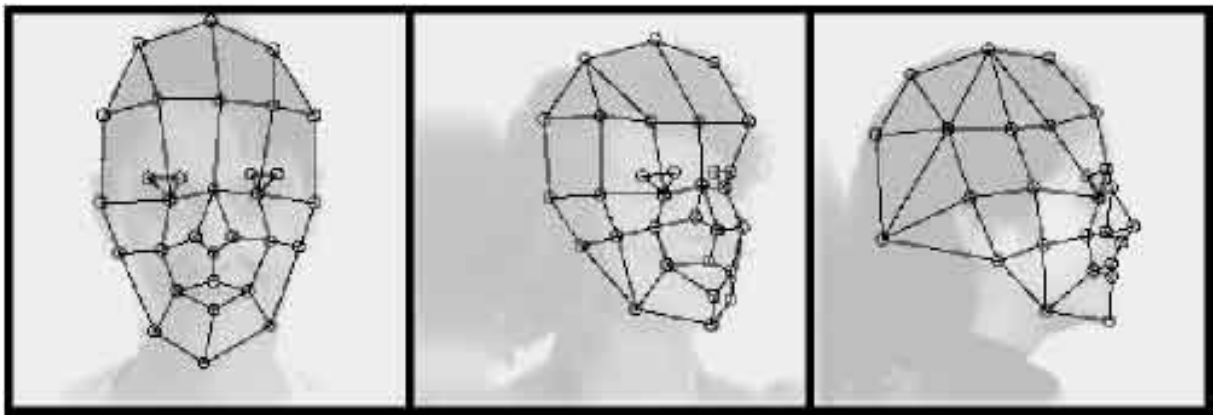


Abb.: 1.6: Beispiele für Bunchgraphen nach [34]

Die Extraktion geometrischer Bildparameter ermöglicht eine Reihe weiterer Anwendungen wie z.B. die Registrierung von Graphiken mittels dieser Parameter. Die Berechnung von Punkten (Knoten) und die sich anschließende Speicherung der entsprechenden Punktgeometrie stellt eine Basisaufgabe aus dem Fachgebiet der Bilderkennung dar. Aus den Punkten lassen sich mit weiterführenden Rechenverfahren dann Linien, Flächen und graphische Objekte generieren.

Das Verfahren der Extrahierung geometrischer Merkmale eignet sich für den Vergleich von Objekten, die weitgehend ähnliche und relativ einfache Geometrien aufweisen. Bei der Anwendung auf eine Gesichtserkennung treffen diese Randbedingungen weitgehend zu. Jede Gesichtsaufnahme ist durch einen Graphen in Form eines Gitternetzes und die für jeden Gitterpunkt geltenden Wavelet-Koeffizienten beschrieben.

Soll ein größeres Objekt, wie z.B. eine vollständig abgebildete Person erkannt werden, dann ist eine Vielzahl von Ausgangsgraphen erforderlich und das Verfahren der Speicherung geometrischer Merkmale wird immer aufwändiger. Außerdem erweist sich die Berechnung des direkten Bildvergleiches in der Zeitebene als grundsätzlich problematisch im Hinblick auf Kameraaufnahmen unter verschiedenen Blickwinkeln, Abständen und Beleuchtungsgraden .

Das in diesem Forschungsbericht beschriebene Erkennungssystem arbeitet grundsätzlich ohne Gitternetz. Gerade im Hinblick auf die Erkennung vermummter Personen mit Überwachungskameras sind Gitternetze zur Parameterextraktion kaum geeignet, denn in diesem Fall existieren im Kamerascan keine verlässlichen Gesichtsmerkmale mehr.

1.5 Demonstration einer Bildparameter-Berechnung

Als anschauliche und bewußt einfach gehaltene Demonstration des Ablaufes einer Bilderkennung mittels der Bildparameter eignet sich die Erkennung einer Spielkarte. Dazu erfaßt eine Kamera z.B. die Herz-Zehn als JPG-Datei und skaliert die Helligkeitswerte jedes einzelnen Pixels auf den Bereich zwischen 0 bis 255. Falls der skalierte Helligkeitswert den Grenzwert von 140 unterschreitet, zählt man die Anzahl der Pixel bis der Helligkeitswert den Grenzwert wieder überschreitet und speichert die Mitte dieses Bereiches in einer boolschen Pixelmatrix ab. Das Testen der Helligkeitswerte erfolgt zunächst zeilenweise und dann spaltenweise. Falls die boolsche Pixelmatrix im zeilenweisen und spaltenweisen Durchlauf an der identischen Stelle den Wert „true“ annimmt, handelt es sich um die Mitte eines graphischen Objektes. Die beiden Stuktur-Matrizen nehmen dann die n Pixelkoordinaten (x_i, y_i) der einzelnen Mittelpunkte zur Weiterverarbeitung auf.

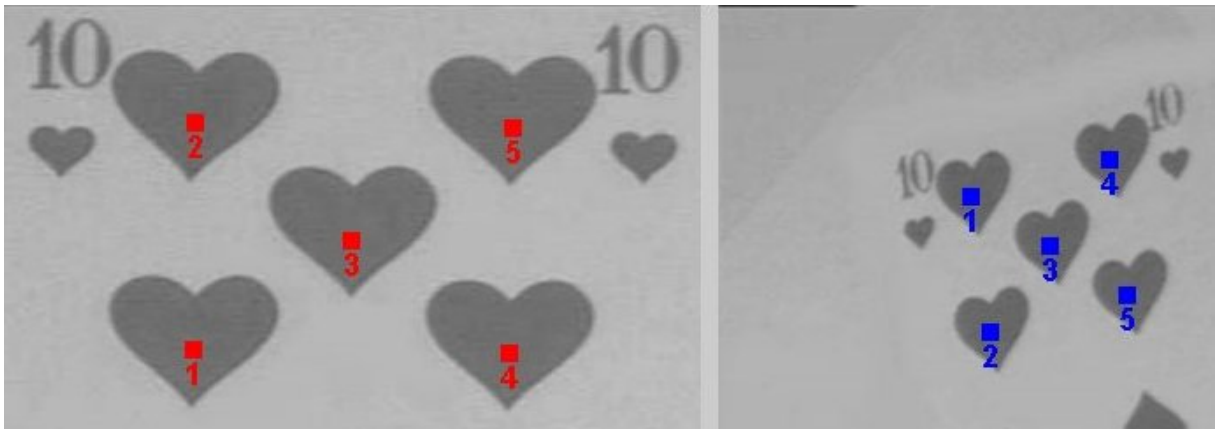


Abb.: 1.7: Gespeicherte und gescannte Aufnahme einer „Herz-Zehn“ Spielkarte

Die Abb. 1.7 stellt im linken Teil den ersten Zwischenschritt der Berechnung dar. Die jeweilige Mitte der geschwärzten Bereiche ist durch eine Pixelmarkierung sichtbar. Der rechte Teil der Abb. 1.7 enthält die entsprechenden Punkte für die gescannte Spielkarte. In diesem vereinfachten Beispiel werden nur die Parameter aus der oberen Bildhälfte berechnet, d.h. die beiden Struktur-Matrizen bestehen jeweils aus 5 Zeilen:

$$Referenz = \begin{pmatrix} 111 & 197 \\ 112 & 68 \\ 201 & 135 \\ 291 & 199 \\ 293 & 71 \end{pmatrix} \qquad Scan = \begin{pmatrix} 144 & 110 \\ 155 & 187 \\ 189 & 138 \\ 223 & 89 \\ 233 & 166 \end{pmatrix} \tag{1.4}$$

Mit Hilfe der Matrizen-Operationen Rotation, Translation und Skalierung wird nun iterativ versucht, die Struktur-Matrix des Referenzbildes mit der Struktur-Matrix des gescannten Bildes möglichst gut in Übereinstimmung zu bringen. Als besondere Schwierigkeit kommt hinzu, dass die Zeilennummern der einzelnen Bildpunkte des Bildscans nicht mehr mit den Zeilennummern des gespeicherten Ausgangsbildes übereinstimmen.

Für das Beispiel der Herz-Zehn mit den 5 Punkten folgen wegen p = n! = 5! insgesamt 120 Permutationen, d.h. es gibt 120 Möglichkeiten die Punktfolge in der Struktur-Matrix des Bildscans zu vertauschen. Die Berechnungen ergaben ein Fehlerminimum bei der Permutation mit der Ordnungsnummer 25 :

$$\begin{array}{cccccc} 1 & 1 & 2 & 3 & 4 & 5 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 23 & 1 & 5 & 2 & 3 & 4 \\ 24 & 2 & 1 & 3 & 4 & 5 \\ 25 & 2 & 1 & 3 & 5 & 4 \\ 26 & 2 & 1 & 4 & 3 & 5 \\ 27 & 2 & 1 & 4 & 5 & 3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 120 & 5 & 4 & 3 & 2 & 1 \end{array} \tag{1.5}$$

Wie aus der Gleichung (1.5) an der Stelle 25 erkennbar ist, tritt das Fehlerminimum dann auf, wenn man die ersten und die letzten beiden Punkte gegenseitig vertauscht. Damit liegt auch die Korrespondenz-Matrix fest:

$$Korrespondenz - Matrix = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{1.6}$$

Die Korrespondenz-Matrix gemäß der Gleichung (1.6) bildet das Vertauschen der Punkte sehr anschaulich ab. Man erkennt auch, dass Ausreisser nicht vorhanden sind, denn die Ordnung der Korrespondenz-Matrix entspricht in beiden Achsen der Anzahl der vorhandenen Punkte. Für den Fall, dass im Bildscan weniger als 4 oder mehr als 6 Punkte auftauchen, nähme die Korrespondenz-Matrix eine rechteckige Form an. Damit folgt dann eine weitgehende Übereinstimmung der berechneten Bildpunkte mit einem Fehlerwert von qmin = 2. Zur graphischen Veranschaulichung des Übergangs der Punkte des Bildscans auf die Punkte des Referenzbildes, visualisiert die Abb. 1.8 nun den schrittweisen „Matching“ - Prozess . Man erkennt deutlich, wie sich die richtige Reihenfolge der Punkte wieder einstellt.

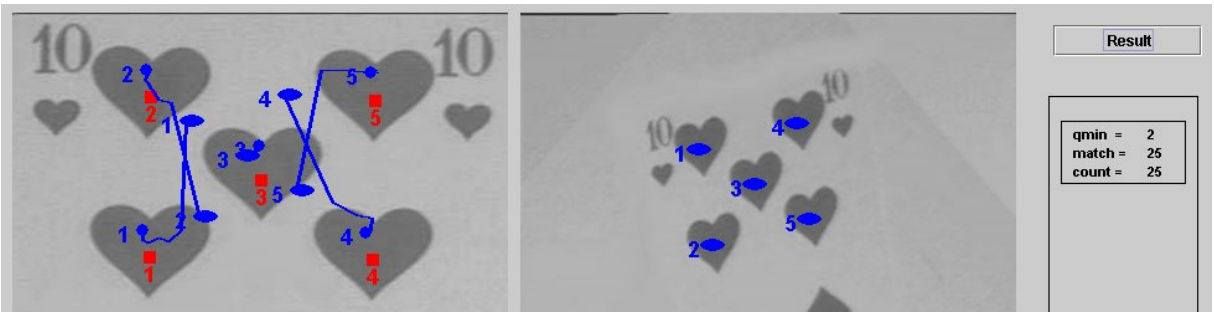


Abb.: 1.8: Matching einer gedrehten „Herz-Zehn“ Spielkarte

Für den Sonderfall zweier unterschiedlicher Spielkarten ergibt sich ein großer Fehler von qmin = 25 zwischen den beiden Struktur-Matrizen, wobei sich dieses Fehlerminimum an der Stelle der zweiten Permutation, bei der nur das mittlere Punktepaar vertauscht ist, einstellt. An diesem Ergebnis fällt auf, dass gegenüber dem Referenzbild nun im Bildscan ein Punkt fehlt. Die Korrespondenz-Matrix (1.7) sieht daher wie folgt aus.

$$Korrespondenz - Matrix = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{1.7}$$

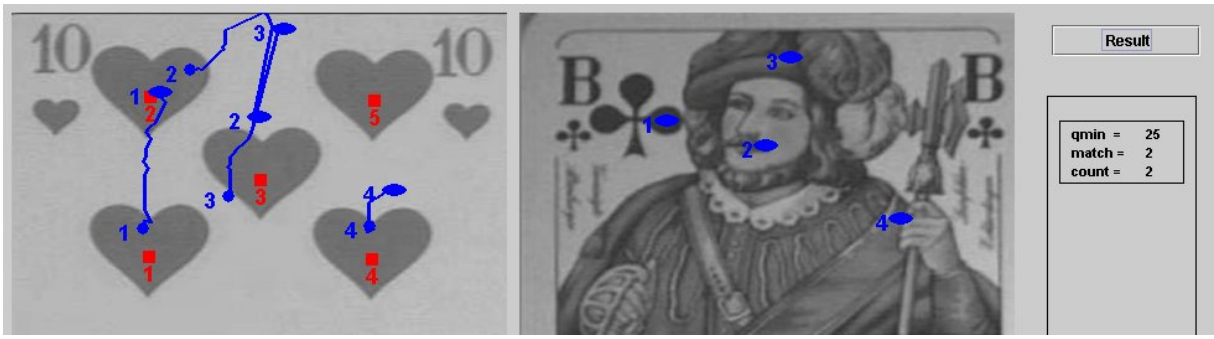


Abb.: 1.9: Matching einer gedrehten „Kreuz-Bube“ Spielkarte

Für das weitere Beispiel unterschiedlicher Objekte, die jedoch eine identische Struktur der graphischen Objekte aufweisen, muss das Matching-Verfahren zwangsläufig versagen. Zur Demonstration eignet sich der Bildscan einer „Pik-Zehn“ Spielkarte, deren graphische Objekte gegenüber einer „Herz-Zehn“ Spielkarte um 180-Grad gedreht sind. Nach Abschluss des Matching-Verfahrens folgt ein minimaler Restfehler von qmin =3, d.h. es wurde fälscherlicherweise eine „Herz-Zehn“ Spielkarte erkannt. Die unterschiedlichen Farben der beiden Spielkarten gingen in die Berechnungen nicht ein.

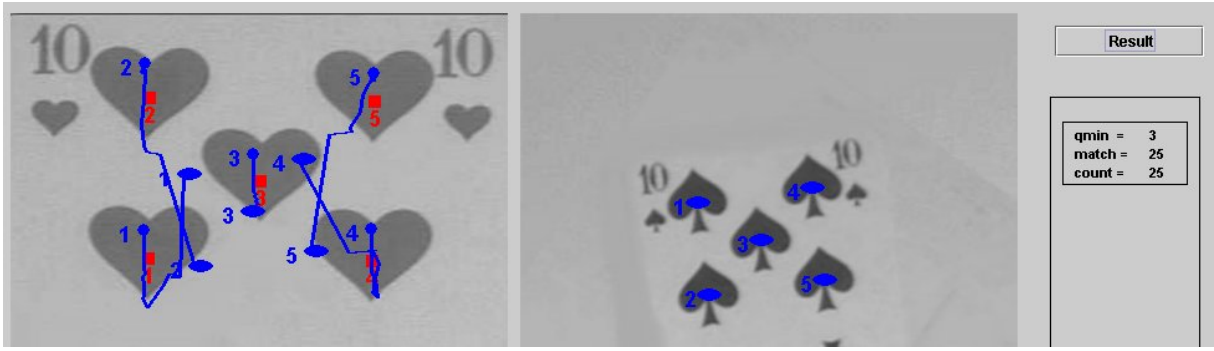


Abb.: 1.10: Matching einer gedrehten „Pik-Zehn“ Spielkarte

Die Normierung der Scan-Matrix und die Bilderkennung erfolgte nach einem iterativen und intuitiven Verfahren, bei dem die ganze Problematik der Gesichtserkennung auf der Basis von Parametern besonders deutlich sichtbar wird. Das eingesetzte Berechnungsverfahren gliedert sich in mehrere mathematischen Operationen, die bis zum Erreichen einer unteren Fehlerschranke iterativ zu wiederholen sind und sich wie folgt beschreiben lassen:

1. Normierung der Mittelpunkts-Koordinaten des Bildscans

   Eine lineare Verschiebung aller Pixelkoordinaten des Bildscans sorgt dafür, dass die Mittelpunkte beider Graphiken übereinstimmen. Die Verschiebung entspricht der Differenz der Mittelpunkts-Koordinaten zwischen den beiden Graphiken. Als Mittelpunkt gilt der Mittelwert aus den jeweils maximalen und mininalen Koordinaten.

2. Skalierung des Bildscans

   Die Multiplikation aller Pixelkoordinaten mit einem Faktor zoom führt dazu, dass der maximale Wert einer x- bzw. y- Koordinate in beiden Graphiken übereinstimmt.

3. Sortieren der Punkte des Bildscans

   Die einzelnen Punkte des Bildscans werden so sortiert, dass zu jedem Punkt des gespeicherten Bildes der Punkt des Bildscans mit dem dazu kürzesten Abstand gehört. Falls der Bildscan mehr Punkte als das gespeichterte Bild enthält, entstehen Ausreisser. Im umgekehrten Fall werden die überzähligen Punkte als Ausreisser des gespeicherten Bildes interpretiert und nicht ausgewertet.

4. Translation

   Ein positiver x-Shift verschiebt den Bildscan nach rechts. Falls die Fehlerquadratsumme zum gespeicherten Bild sich dadurch reduziert, bleibt diese Verschiebung x erhalten. Im anderen Fall testet man eine negative Verschiebung x. Anschließend wiederholt man dieses Probierverfahren für die y-Komponenten.

5. Rotation

   Die Einstellung der optimalen Ausrichtung der Spielkarte erfolgt auch hier wieder mit einem rekursiven Verfahren. Zunächst dreht man den Bildscan um einen kleinen positiven und negativen Winkelwert nach beiden Seiten und stellt fest, welche der Drehungen zur geringeren Differenz zwischen dem Referenzbild und dem Bildscan führt. Falls sich der Fehlerwert in beiden Fällen erhöht, gelangt keine der beiden Drehungen zur Ausführung. Im anderen Fall dreht man den Bildscan in jene Richtung, welche den kleineren Fehler ergab. Durch die Wiederholung dieser Rotationen findet man die günstigste Einstellung für die Ausrichtung des Bildscans heraus.

6. Bilderkennung

   Die Schritte 1 bis 5 wiederholt man für alle gespeicherten Bilder. Als erkannt gilt ein Bild, wenn die Fehlerquadratsumme ein absolutes Minimum erreicht.

Kapitel 2

Aufbereitung der Bilddaten

2.1 Komprimierung mit Wavelets

In der Regel erweist sich die Anzahl der Bildpunkte vor dem Hintergrund der Echtzeitfähigkeit eines Bilderkennungsverfahrens meistens als viel zu umfangreich. Ein farbiges Kamerabild von z.B. 640x480 Pixeln erfordert für jedes Pixel 4 Byte Speicherplatz. Insgesamt folgt daraus eine Datenmenge von 640x480x4 Byte, was ca. 1,3 MByte entspricht. Geht man auf Grauwerte über, dann reduziert sich die Datenmenge um ein Viertel auf ca. 300 kByte. Allerdings ist dieses Pixelbild, das man am besten in einer Matrix vom Typ Byte abspeichert, in der Regel immer noch zu umfangreich.

Zur Reduktion der Bilddaten stehen eine Vielzahl von Verfahren zur Verfügung. Letzlich kommt es im Zusammenhang mit der nachfolgend durchzuführenden Bilderkennung darauf an, trotz der notwendigen Datenreduktion, den Informationsgehalt des Bildes weitgehend zu erhalten. Im Gegensatz zu vielen Standardverfahren der Bilddaten-Kompression, bei denen das Leistungsvermögen des menschlichen Auges die Kompressionsrate begrenzt, geht es bei dem hier anzustrebenden maschinellen Bilderkennen darum, den nachfolgenden Algorithmen einen möglichst hohen Informationsgehalt anzubieten. Die menschliche Sehleistung und das subjektive Empfinden beim Betrachten eines Bildes spielt in diesem Zusammenhang keine Rolle.

Bei dem Wavelet-Verfahren handelt es sich um eine spezielle Integraltransformation, die im Zusammenhang mit der Bildverarbeitung eine große Bedeutung erlangt hat. Anschaulich betrachtet leistet eine Integraltransformation die Umwandlung pixelbasierter Bilddaten in reelle oder komplexe Koeffizientensätze. Bei Verwendung geeigneter Transformationsfunktionen, dem „Transformationskern“, erzielt man eine spürbare Datenreduktion, da im günstigsten Fall wenige Koeffizienten ausreichen, das Orginalbild näherungsweise fehlerfrei zu beschreiben. Zur Berechnung des Bildes aus den Koeffizientensätzen dienen dann die Gleichungen zur Rücktransformation. Grundsätzlich betrachtet, bezahlt man die Speicherreduzierung mit einer Erhöhung des Rechenaufwandes für die Hin- und Rücktransformation.

Wavelets, die bereits seit 1910 durch den Mathematiker Haar bekannt sind und ab 1986 zur Bildverarbeitung eingesetzt wurden ([29], [41],[42]) stellen die Weiterentwicklung der bekannten Fourier-Transformationen dar. Während bei Fourier als Kern der Integraltransformation periodische Sinus- und Cosinus-Funktionen dienen, wählt man bei der Wavelet-Transformation ganz spezielle, kleine wellenförmige Schwingungen („wavelets“) , welche den folgenden 3 Bedingungen genügen müssen:

1. Die Fläche unter einer Wavelet-Funktion ist Null.
2. Der Defintionsbereich ist begrenzt.
3. Wavelet-Funktionen hängen von zwei Parametern ab, mit denen der Funktionsverlauf skaliert und linear verschoben werden kann.

Im einfachen Fall der Haar-Wavelets, die jedoch für digitale Signale von großer Bedeutung sind, basiert die Wavelet-Funktion $\Psi(a,b)$ auf einer einzelnen symmetrischen Rechteckschwingung f(x), deren Ausdehnung und Lage auf der horizontalen x-Achse die beiden reellen Parameter a und b definieren.

$$\text{f(x)} = \begin{cases} 1 & \text{für} \quad 0 \leq x < 0.5 \\ -1 & \text{für} \quad 0.5 \leq x < 1.0 \\ 0 & \text{sonst} \end{cases} \tag{2.1}$$

$$\Psi(a,b) = \frac{1}{\sqrt{|a|}} \quad f(\frac{x-b}{a}) \qquad \text{mit a,b reell und a} \neq 0 \tag{2.2}$$

Für die diskrete Wavelet-Transformation ersetzt man die Parameter durch ganzzahlige Zähler i und j und erhöht die Parameter im dualen Zahlensystem:

$$\Psi_{i,j}(x) = \frac{1}{\sqrt{2^i}} \quad f(\frac{x - j \cdot 2^i}{2^i}) = \frac{1}{\sqrt{2^i}} \quad f(2^{-i} \cdot x - j) \tag{2.3}$$

Der Parameter i in der Gleichung (2.3) streckt und der Parameter j verschiebt das Wavelet. Strecken bedeutet, dass die Rechteck-Schwingung mittels i in der Breite einstellbar ist. Durch die Verschiebung j wandert das Wavelet auf der x-Achse nach rechts. Damit ist dieses Wavelet den starren Frequenzlinien einer Fourier-Transformation vor allem bei unterschiedlichen Frequenzbereichen in verteilten Zeitabschnitten des Ausgangssignals weit überlegen, d.h. es genügen wenige Koeffizienten um ein Zeitsignal darzustellen.

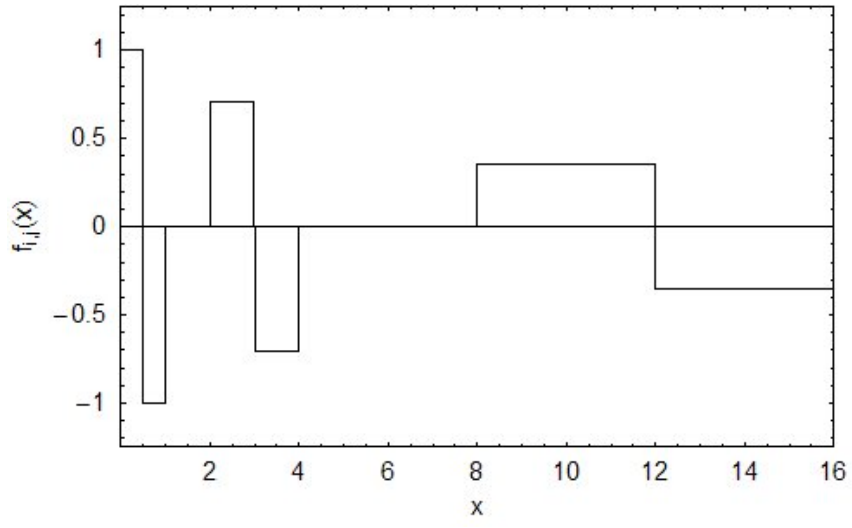


Abb.: 2.1: Haar-Wavelets (links: i=j=0) , (Mitte links: i=1,j=1) und (rechts: i=3, j=1)

Zur Bildkompression mittels Wavelet-Koeffizienten eignet sich am besten die „fast discrete wavelet transform“, welche einen erheblich schnelleren Algorithmus zur Berechnung der Koeffizienten anbietet und 1996 von W. Sweldens [39] entwickelt wurde. Ausgehend von zwei aufeinander folgenden Graustufenwerten a und b einer Pixelzeile bzw. Pixelspalte gilt:

$$\boxed{\text{Hin - Transformation:} \quad s = \frac{a+b}{2} \quad \text{bzw.} \quad d = \frac{a-b}{2}} \tag{2.4}$$

$$\boxed{\text{Rück - Transformation:} \quad a = s - d \quad \text{bzw.} \quad b = s + d} \tag{2.5}$$

Die Verarbeitung der Zahlenwerte erfolgt immer paarweise. Aus z.B. 8 Zahlenwerten ergeben sich also 4 s- und 4 d-Werte. Die s-Werte nehmen die ersten 4 Plätze und die d-Werte die Plätze 5 bis 8 der neuen Zahlenfolge ein. Eine Datenreduktion stellt sich ein, wenn d-Werte, die unterhalb eines Schwellwertes liegen, entfallen. In einem Wiederholungsschritt entstehen anschließend aus dem Wertepaar s und d neue Summen und Differenzen. Falls noch alle d-Werte vorhanden sind, läßt sich das Orginalsignal mittels der Gleichungen zur Rücktransformation fehlerfrei regenerieren. Selbstverständlich muss die Anzahl der zu transformierenden Zahlenwerte so oft durch 2 teilbar sein, wie die Anzahl der Wiederholungen dies erfordert. Führt man bei den 8 Zahlenwerten 3 Reduktionsschritte durch, dann bleiben nur noch ein s-Wert und 7 d-Werte übrig.

Beispiel 2.1: Gegeben sei eine Pixelfolge aus 8 Werten:

gegebene Pixelfolge	$\{56, 40, 8, 24, 48, 48, 40, 16\}$
halbierte s-Folge	$\{48, 16, 48, 28\}$
halbierte d-Folge	$\{8, -8, 0, 12\}$
Rücktransformation	$\{56, 40, 8, 24, 48, 48, 40, 16\}$

Zur Komprimierung von Pixelbildern transformiert man zunächst Zeile für Zeile gemäß der s-Formel (Gleichung 2.4). Das dadurch enstehende neue Halbbild weist n/2 Spalten und n Zeilen auf. Dann wiederholt man den Algorithmus spaltenweise und erhält ein Viertelbild mit n/2 Zeilen und n/2 Spalten. Zur Veranschaulichung wurde die zweite Zeile der Pixelmatrix ebenfalls mit Zahlenwerten besetzt. Zur Bestimmung von Umrissen führt man diese Berechnungen mit der Differenz d durch.

$$\text{Bild} = \begin{Bmatrix} 56 & 40 & 8 & 24 & 48 & 48 & 40 & 16 \\ 40 & 20 & 10 & 10 & 5 & 15 & 8 & 20 \\ x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x \end{Bmatrix} \rightarrow \text{Halbbild} = \begin{Bmatrix} 48 & 16 & 48 & 28 \\ 30 & 10 & 10 & 14 \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{Bmatrix} \tag{2.6}$$

$$\rightarrow \text{Viertelbild} = \begin{Bmatrix} 39 & 13 & 29 & 21 \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{Bmatrix} \tag{2.7}$$

Beispiel 2.2:

Die Abb. 2.2 stellt links oben ein Orginalbild des Autors dar und dazu in der rechten oberen Bildhälfe das Ergebnis einer spaltenweisen Wavelet-Operation. Die Anzahl der Spalten des Bildes halbiert sich und es entsteht daher eine verzerrte Darstellung. Nach einer weiteren zeilenweisen Wavelet-Operation erhält man das links unten dargestellte Ergebnis, bei dem das Seitenverhältnis wieder stimmmt und bei dem sich die Anzahl der Pixel auf ein Viertel reduziert hat. Die Farbinformation des Orginalbildes bildet man in den Mittelwert aus den drei RGB-Farbwerten ab.

Die Darstellung in der Mitte der unteren Bildhälfte der Abb. 2.2 zeigt das Differenzbild, in dem alle Grauwerte, die den Schwellwert unterschreiten, auf Weiß gesetzt werden. Man erkennt, wie dadurch die Konturen sichtbar werden. Rechts unten stellt die Abb. 2.2 das Differenzbild ohne diese Grenzwertabfrage dar. Ein starkes Rauschen überlagert nun die Darstellung, d.h. mit der eigentlichen Differenz-Darstellung lässt sich kaum etwas anfangen. Die Wertefolge d dient einzig dazu, bei der Wiederherstellung des Bildes zusammen mit der Wertefolge s, das Orginalbild wieder zu berechnen (siehe Gleichung 2.5). Diese Aufgabe ist jedoch im Zusammenhang mit der hier betrachteten Bildererkennung nicht erforderlich und nur dann sinnvoll, wenn es darum geht, Bilddaten zu komprimieren, zu übertragen und beim Empfänger wieder darzustellen. Für die wichtige Aufgabe der Bestimmung der Randlinien eines graphischen Objektes eignen sich die im folgenden Abschnitt beschriebenen speziellen Filterverfahren erheblich besser.

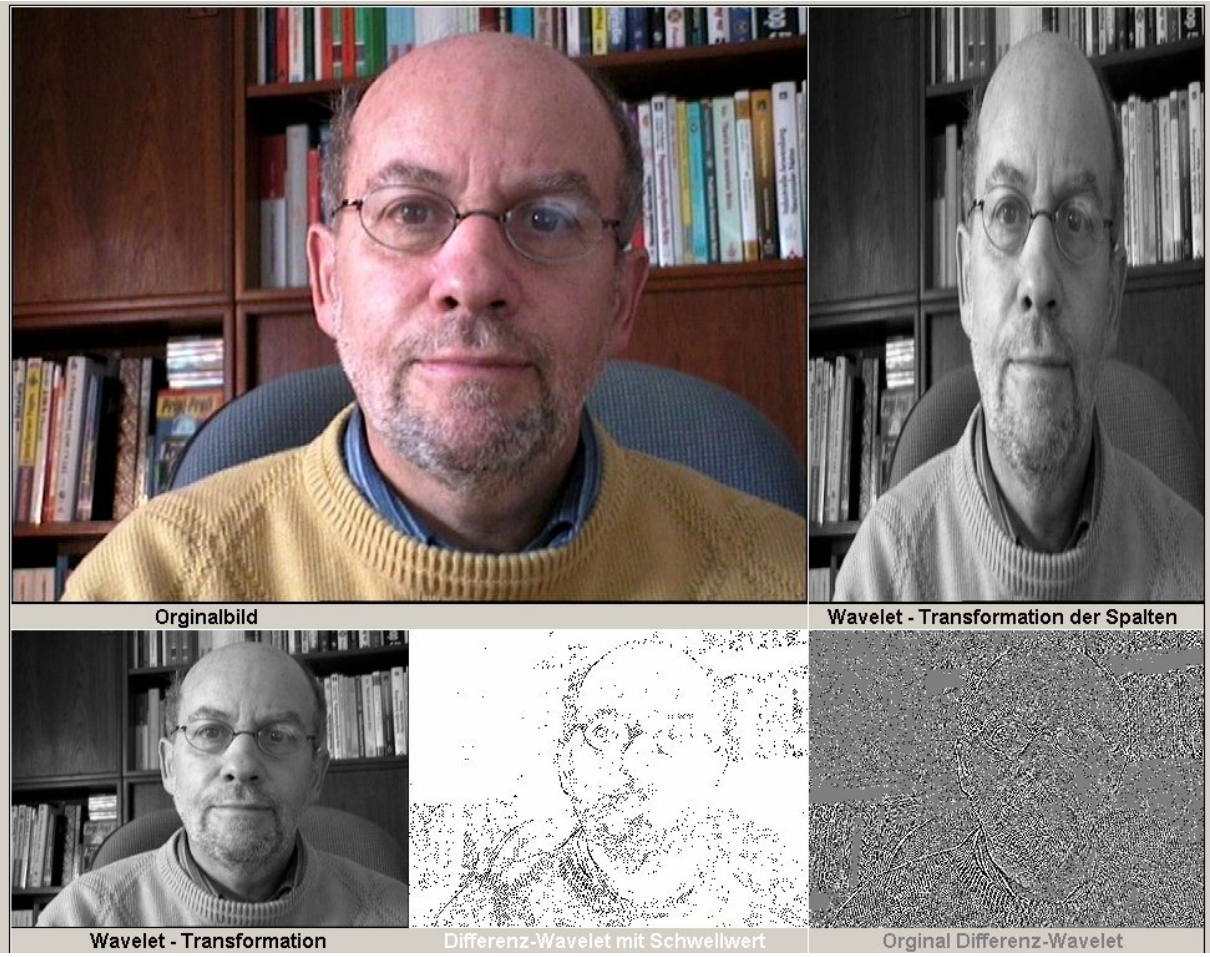


Abb.: 2.2: Wavelet - Bildfolge

2.2 Transformation der Pixelwerte

Gesichtsbilder, die von einer Kamera erfasst werden, können sehr unterschiedliche Helligkeit besitzen und durch kleine, große oder verstreute Wertebereiche für die Farbwerte gekennzeichnet sein. Wünschenswert wären jedoch normierte, kontrastreiche und gleichverteilte Pixelwerte, die den vorgegebenen Wertebereich vollständig ausnutzen und damit den nachfolgenden Algorithmen der Bilderkennung optimale Voraussetzungen bieten. Das Fachgebiet der Bildbearbeitung kennt eine Vielzahl von Methoden, mit denen sich die gegebenen Pixelwerte in weiten Bereichen und mit den unterschiedlichsten Zielsetzungen verändern lassen.

Mit Blick auf eine möglichst weitreichende Unterstützung der Methoden der Gesichtserkennung, die im Gegensatz zur Bildbearbeitung nicht primär auf eine qualitative Verbesserung der graphischen Darstellung abzielen, erweist sich eine Grauwert-Transformation als unverzichtbar.

Bei einer Grauwert-Transformation, die anschaulich betrachtet einer Spreizung der Pixelwerte entspricht, skaliert man die einzelnen Pixelwerte so, dass ihr möglicher Zahlenbereich ganz ausgenutzt wird. Für den Fall einer 8-Bit Darstellung wäre dies der Bereich von 0 bis 255. Speichert man in der Programmiersprache Java die Pixelwerte in einem Feld vom Typ byte ab, dann muss sich der mögliche Zahlenbereich zwischen ± 127 erstrecken. Der ursprüngliche mininale Pixelwert g_{min} nimmt dann den Wert -127 an und der entsprechende maximale Pixelwert g_{max} erreicht den Wert 127.

Häufig ist es im Anwendungsfall einer Bilderkennung günstiger, die Pixelwerte nicht stetig zu transformieren, sondern die Ausgangswerte g_{output} stufenartig in Bereiche zu unterteilen, d.h eine Quantisierung der Pixelwerte zu realisieren.

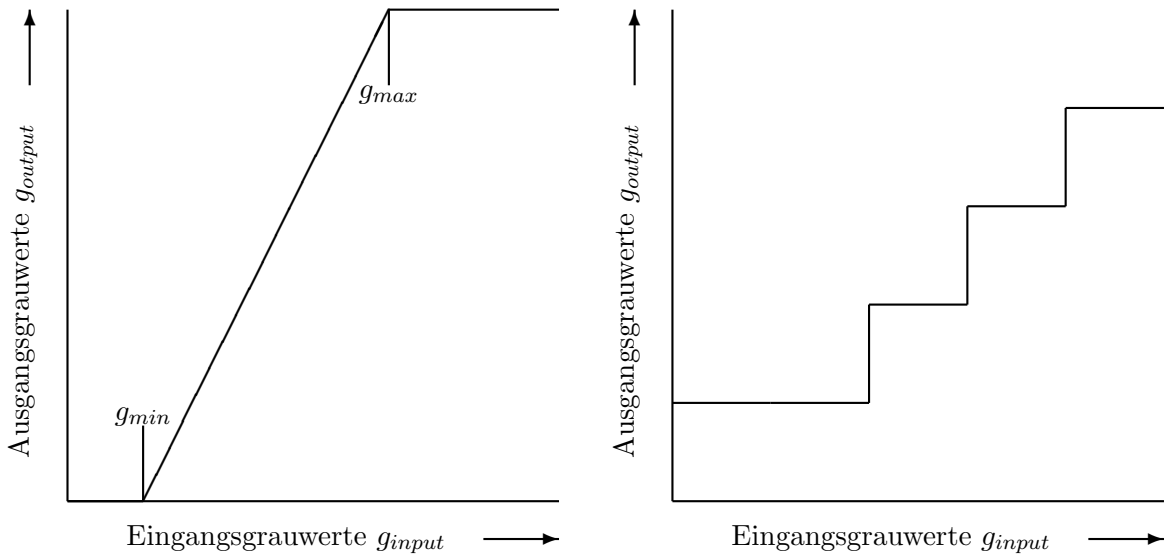


Abb.: 2.3: Beispiele für Grauwert-Spreizungen

Die Gleichung (2.8) beschreibt den linearen Zusammenhang zwischen den Eingangsparametern g_{min} und g_{max} und den Ausgangsparametern $g_{minimum}$ und $g_{maximum}$ nach der Spreizung der Grauwerte von g_{input} auf g_{output}. Vor der eigentlich Anwendung der Gleichung (2.8) muss die Bestimmung des minimalen und des maximalen Pixelwertes g_{min} bzw. g_{max} des zu transformierenden Bildes erfolgen. Die Berechnung der stufenförmigen Pixelwerte läßt sich programmiertechnisch mit einer Rundungsoperation (2.9) darstellen:

$$g_{output} = (g_{input} - g_{min}) \frac{g_{maximum} - g_{minimum}}{g_{max} - g_{min}} - g_{minimum} \tag{2.8}$$

$$g_{output} = \text{Math.round} \left(\frac{g_{output}}{\text{Quantisierungswert}} \right) \cdot \text{Quantisierungswert} \tag{2.9}$$

Für den Sonderfall Quantisierungswert = 10 ergeben sich folgende Ausgangswerte

$$g_{output} = \{-120, -110, \cdots, -10, 0, 10, \cdots, 110, 120\}$$

Die Abbildung 2.3 zeigt die beiden Kennlinien der Grauwert-Transformationen. Grundsätzlich ist erst die in der linken Bildhälfte und dann die in der rechten Bildhälfte der Abb. 2.3 beschriebene Transformation auf das Ergebnis der stetigen Grauwert-Transformation anzuwenden.

Ein Beispiel für die Anwendung einer Pixel-Transformation stellt die Abb. 2.4 im oberen linken Bildausschnitt des Ausgangsbilds der Wavelet-Transformation dar. Man erkennt im oberen rechten Bildausschnitt, welche deutliche Bildverbesserung die Grauwert-Transformation ermöglicht. Die beiden unteren Bildausschnitte der Abb. 2.4 stellen die Ergebnisse für quantisierte Pixelwerte dar. Im Bildausschnitt unten rechts sieht man, dass nun die Grauwerte nur noch durch die vier Bereiche gegeben sind.

Abb.: 2.4: Transformation der Pixelwerte

2.3 Kantenverstärkung

Für jene Methoden der Bilderkennung, welche sich an den Umrissen eines Gesichtes orientieren, ist es erforderlich, die Kanten der Gesichtsaufnahme zu verstärken. Dafür stehen aus dem Bereich der Bildverarbeitung eine Reihe von Verfahren zur Verfügung. Jede Kantendetektion setzt zunächst die Verstärkung der Kanten voraus, wobei es immer darum geht, vorhandene Pixelwert-Änderungen zu verstärken. Durch die Berechnung von Pixelwert-Differenzen in waagrechter und senkrechter Richtung erhält man Differenzen, die am Ort einer Bildkante deutlich von Null verschieden sind. Eine nachgeschaltete Schwellwert-Operation sorgt dann dafür, dass nur die größeren Differenzen als erkannte Kanten interpretiert werden.

Für den Anwendungsfall einer Bilderkennung sind jene Verfahren der Kantenverstärkung sinnvoll, welche die beiden folgenden Bedingungen erfüllen:

- Geringer Rechenzeitbedarf
- Zusätzliche Parameter zur optimalen Einstellung der Kantenverstärkung

Die einfache Gleichung (2.10) erfüllt beide Bedingungen. Ausgehend vom Mittelwert der Pixelwerte pm(i,j) in einem z.B. quadratischen Bildausschnitt mit n^2 Pixeln, die zentriert um den momentan betrachteten Punkt (i,j) liegen, ermittelt die Gleichung (2.10) für den Mittelpunkt dieses Bildausschnittes die Differenz zwischen dem zentralen Pixelwert und dem Mittelwert des Bildausschnittes. Mit dem Parameter α ist die Verstärkung dieser Differenz beliebig einstellbar.

$$p(i,j) = \frac{p(i,j) - \alpha\ p_m(i,j)}{1-\alpha}\ ; \quad 0 < \alpha < 1 \tag{2.10}$$

$$p_m(i,j) = \frac{1}{n^2} \sum_{u=1}^{n} \sum_{v=1}^{n} p(i+u, j+v) \tag{2.11}$$

Eine alternative und sehr bekannte Methode der Kantenverstärkung besteht in der Anwendung von Filteralgorithmen, die z.B. einen Sobel- , Prewitt- oder Laplace-Filterkern besitzen. Der Laplace-Filterkern der Gleichung (2.12) bietet die Möglichkeit, durch einen zusätzlichen Parameter ϵ das Verhalten der Kantenverstärkung zu optimieren. Außerdem führt dieser Filterkern eine richtungsunabhängige Kantenverstärkung durch, was im Vergleich zu anderen Filterkernen die Rechenzeit halbiert. Der Algorithmus geht ebenfalls von einem Bildausschnitt aus, der jedoch exakt 3 x 3 Pixel enthalten muss. Die Pixelwerte in diesem Bildausschnitt werden mit den entsprechenden Matrizenelementen der quadratischen 3x3 Filterkern-Matrix gewichtet aufaddiert.

$$\text{Laplace-Filterkern Lp=} \begin{pmatrix} 0 & -\epsilon & 0 \\ -\epsilon & 1+4\cdot\epsilon & -\epsilon \\ 0 & -\epsilon & 0 \end{pmatrix} \tag{2.12}$$

$$p(i,j) = \frac{1}{n^2} \sum_{u=1}^{n} \sum_{v=1}^{n} p(i+u, j+v) \cdot Lp(u,v) \tag{2.13}$$

Das Ergebnis überschreibt den zentralen Pixelwert des Bildausschnitts. Die Gleichung (2.13) beschreibt die Filterung in mathematischer Form:

Die Abb. 2.5 und Abb. 2.6 zeigen am Beispiel quantisierter Gesichtsaufnahmen die Wirkungsweise beider Algorithmen zur Kantenverstärkung. Durch die Einstellung der entsprechenden Parameter α bzw. ϵ lassen sich bei relativ geringer Rechenzeit optimale Ergebnisse erzielen.

Abb.: 2.5: Verstärken der Bildkanten am Beispiel 1

Optimal bedeutet in diesem Zusammenhang, dass für die nachfolgenden Methoden der Bildererkennung numerisch günstige und informative Pixelwerte entstehen.

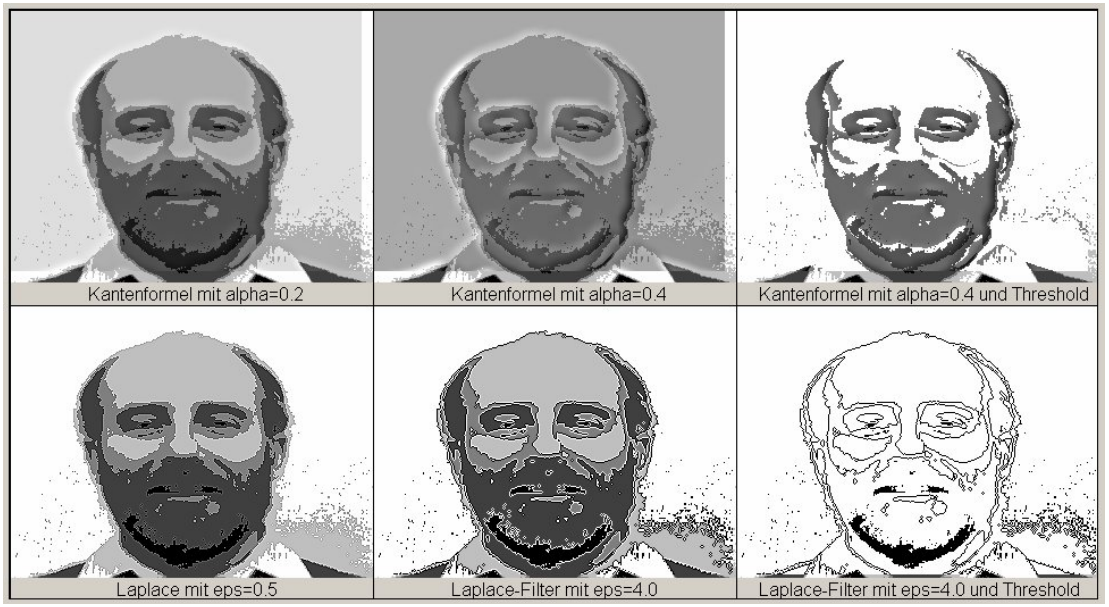


Abb.: 2.6: Verstärken der Bildkanten am Beispiel 2

2.4 Zentrieren eines Objekts

Bei der Erfassung einer Person, die sich an einer Kamera vorbei bewegt, liegt das Gesicht als graphisches Objekt in der Regel außerhalb der Bildmitte. Vergleicht man das auf diese Weise gescannte Bild mit den gespeicherten Aufnahmen, dann ist auf der Ebene der eigentlichen Bilderkennung diese örtliche Verschiebung zu kompensieren.

Viel günstiger ist es dagegen, die Position der graphischen Objekte schon vorab bei der Bildaufbereitung abzugleichen und die Kompensation der restlichen Verschiebung den Erkennungsalgorithmen zu überlassen.

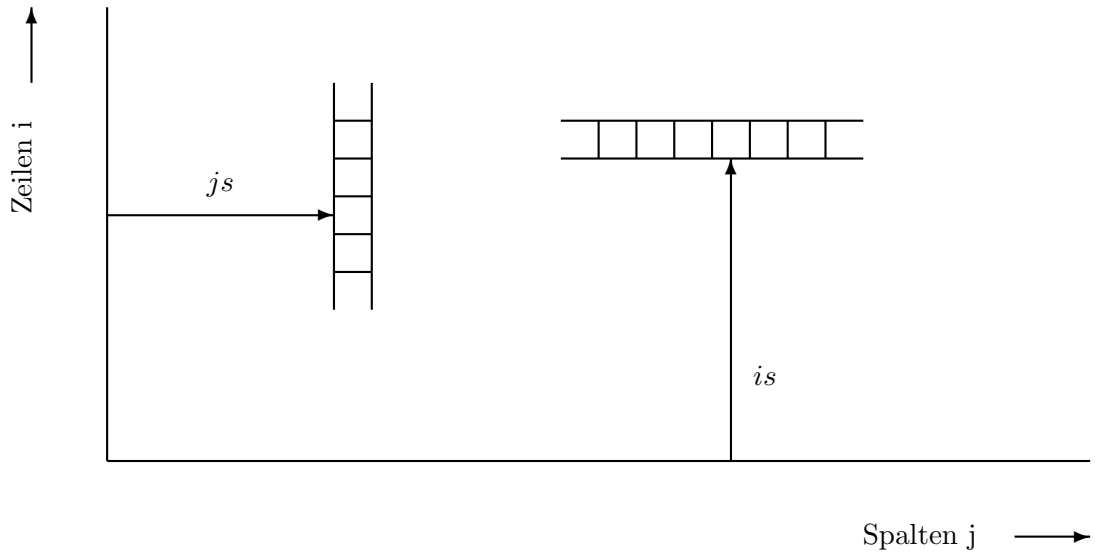


Abb.: 2.7: Grundprinzip einer Flächen-Schwerpunktsberechnung

Zur Zentrierung eines Bildes berechnet man den Schwerpunkt der Bildfläche. Das Gewicht eines Flächenelementes entspricht dem jeweiligen Pixelwert, der mit seinem Abstand zur waagrechten bzw. senkrechten Achse multipliziert und aufaddiert wird. Die Summe aller Momente ergibt sich aus dem Mittelpunktsabstand multipliziert mit dem Gewicht der gesamten Fläche. Der physikalische Grundgedanke besagt, dass sich eine Fläche im Gleichgewicht befindet, wenn man sie im Mittelpunkt abstützt.

Abb.: 2.8: Zentrieren eines Gesichtsbildes

Die Abb. 2.8 zeigt am Beispiel einer Gesichtsaufnahme, wie auf der Basis einer Schwerpunktsberechnung das graphische Objekt zentriert werden kann. In dem vorliegenden Beispiel verschob sich das Ausgangsbild um 18 Pixel nach oben und um 55 Pixel nach links.

Die Gleichungen (2.14) bis (2.18) beschreiben die entsprechenden Berechnungsverfahren.

$$mi = Pixelsumme \cdot is = \sum_{i=1}^{imax} \left[i \sum_{j=1}^{jmax} Pixelwert(i,j) \right] \tag{2.14}$$

$$mj = Pixelsumme \cdot js = \sum_{j=1}^{jmax} \left[j \sum_{i=1}^{imax} Pixelwert(i,j) \right] \tag{2.15}$$

$$Pixelsumme = \sum_{i=1}^{imax} \sum_{j=1}^{jmax} Pixelwert(i,j) \tag{2.16}$$

$$is = \frac{mi}{Pixelsumme} \tag{2.17}$$

$$js = \frac{mj}{Pixelsumme} \tag{2.18}$$

Bei der Zentrierung ist zu beachten, dass eigentlich nur jene Pixel, welche das eigentliche graphische Objekt kennzeichnen, zur Berechnung der Schwerpunkt-Koordinaten verwendet werden dürfen. In der Abb. 2.8 erfolgten die Berechnungen nur mit den negativen Pixelwerten der Byte-Darstellung. Ein ausgeprägter schwarzer Bildbereich, z.B. in der oberen Bildhälfte, verschiebt den Schwerpunkt in die falsche Richtung. Deshalb stellt die Zentrierung des Bildinhaltes nach der Grauwert-Transformation und der Kantenverstärkung den letzten Schritt der Bildaufbereitung dar.

In der Abb. 2.9 gelingt es durch eine entsprechende Pixeltransformation und Kantenverstärkung, die fehlgeschlagene Zentrierung der rechten Bildhälfte zu überwinden. Die rechte Bildhälfte der Abb. 2.9 zeigt die gelungene Zentrierung.

Abb.: 2.9: Zentrieren eines Gesichtsbildes vor und nach der Bildaufbereitung

Grundsätzlich sei an dieser Stelle daran erinnert:

Bei der Bilderkennung kommt es nicht auf die Schönheit oder die Realitätstreue einer Darstellung an, sondern darauf, dass die Bilderkennungsalgorithmen mit optimal skalierter Numerik und mit informativen Darstellungen der zu erkennenden graphischen Objekte angesteuert werden.

Kapitel 3

Die Eigenwert-Methode

3.1 Grundlagen

Der Ausdruck Eigenvektor weist auf die Eigenwert-Methode aus dem mathematischen Gebiet der linearen Algebra hin und beschreibt ein Verfahren, welches die Objekterkennung mittels Eigenvektoren ermöglicht. Eigenvektoren bzw. Eigenmatrizen setzt man ein, um eine quadratische Matrix A der Ordnung n auf die Diagonalform B mittels der zu bestimmenden Transformationsmatrix T zu transformieren. Dann löst man das gegebene mathematische Problem mittels der Diagonalmatrix B, bei der nur die Hauptdiagonale mit Zahlenwerten, den sogenannten Eigenwerten λ, besetzt ist. Die erhaltene Lösung muss abschließend wieder zurücktransformiert werden.

$$B = T^{-1}AT = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \lambda_n \end{pmatrix} \quad \longrightarrow \quad A = T \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \lambda_n \end{pmatrix} T^{-1} \tag{3.1}$$

Geometrisch lässt sich die Eigenwert-Methode am Beispiel einer gedrehten Ellipse veranschaulichen. Durch die Transformation der Ellipse in den Vektorraum der Eigenvektoren dreht sich die Ellipse in die Hauptachsen. Zur Erkennung von Ellipsen könnte man daher den Bildvergleich im Eigenvektor-Raum durchführen. Der besondere Vorteil der Eigenwertmethode besteht darin, dass die Orientierung des gescannten Bildes nicht bekannt sein muss.

Die Berechnung der Eigenwerte aus denen dann die Eigenvektoren und schließlich die Transformationsmatrix folgen, geht von den folgenden Definitionsgleichungen aus:

$$A\,x = \lambda\,x \tag{3.2}$$

$$(A - \lambda\,E)\,x = 0 \quad \rightarrow \quad \det(A - \lambda\,E) = 0 \tag{3.3}$$

$$\det(A - \lambda\,E) = \begin{vmatrix} a_{11} - \lambda & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} - \lambda & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} - \lambda \end{vmatrix} = 0 \tag{3.4}$$

A bezeichnet die quadratische Ausgangsmatrix der Ordnung n und jeder Vektor x, der die Gleichung erfüllt, heißt Eigenvektor. E stellt die Einheitsmatrix dar. Das zweistufige Lösungsverfahren erfordert zunächst die Berechnung der n Eigenwerte aus der Determinantengleichung (3.4). Mit den berechneten Eigenwerten bestimmt man dann mittels des linearen Gleichungssystems (3.2) die Eigenvektoren für jeden einzelnen Eigenwert.

Beispiel 3.1:

1. Berechnung der Eigenwerte

$$A = \begin{pmatrix} 5 & 4 \\ 1 & 2 \end{pmatrix} \quad \longrightarrow \quad det(A - \lambda E) = \begin{vmatrix} 5-\lambda & 4 \\ 1 & 2-\lambda \end{vmatrix}$$
$$\longrightarrow \quad (5-\lambda)(2-\lambda) - 4 = 0 \quad \longrightarrow \quad \lambda^2 - 7\lambda + 6 = 0$$
$$\longrightarrow \quad \lambda_1 = \frac{7}{2} + \sqrt{\frac{49}{4} - 6} = 6$$
$$\longrightarrow \quad \lambda_2 = \frac{7}{2} - \sqrt{\frac{49}{4} - 6} = 1$$

2. Berechnung des v_1 - Eigenvektors

$$\begin{pmatrix} a_{11} - \lambda_1 & a_{12} \\ a_{21} & a_{22} - \lambda_1 \end{pmatrix} v_1 = 0 \quad \longrightarrow \quad \begin{pmatrix} 5-6 & 4 \\ 1 & 2-6 \end{pmatrix} \begin{pmatrix} v_{1x} \\ v_{1y} \end{pmatrix} = 0$$

$I : -v_{1x} + 4v_{1y} = 0 \qquad II : -v_{1x} - 4v_{1y} = 0$

Die Gleichungen I und II sind grundsätzlich linear abhängig!

$$\frac{v_{1x}}{v_{1y}} = \frac{4}{1} \quad \longrightarrow \quad v_1 = \frac{1}{\sqrt{17}} \begin{pmatrix} 4 \\ 1 \end{pmatrix}$$

3. Berechnung des v_2 - Eigenvektors

$$\begin{pmatrix} a_{11} - \lambda_2 & a_{12} \\ a_{21} & a_{22} - \lambda_2 \end{pmatrix} v_2 = 0 \quad \longrightarrow \quad \begin{pmatrix} 5-1 & 4 \\ 1 & 2-1 \end{pmatrix} \begin{pmatrix} v_{2x} \\ v_{2y} \end{pmatrix} = 0$$

$I : 4v_{2x} + 4v_{2y} = 0 \qquad II : v_{2x} + v_{2y} = 0$

Die Gleichungen I und II sind wieder - wie erwartet - linear abhängig.

$$\frac{v_{2x}}{v_{2y}} = \frac{-1}{1} \quad \longrightarrow \quad v_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

4. Bestimmung der Transformationsmatrix aus den Eigenvektoren

$$T = (v_1, v_2) = \begin{pmatrix} \frac{4}{\sqrt{17}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{17}} & \frac{1}{\sqrt{2}} \end{pmatrix} \quad \longrightarrow \quad T^{-1} = \frac{1}{\frac{4}{\sqrt{34}} + \frac{1}{\sqrt{34}}} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{17}} & \frac{4}{\sqrt{17}} \end{pmatrix}$$

5. Ähnlichkeitstransformation (Kontrollrechnung)

$$B = \frac{1}{\frac{5}{\sqrt{34}}} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{17}} & \frac{4}{\sqrt{17}} \end{pmatrix} \begin{pmatrix} 5 & 4 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} \frac{4}{\sqrt{17}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{17}} & \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 6 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

3.2 Bilderkennung mittels der Eigenwert-Methode

Wendet man die Eigenvektor-Methode auf eine Gesichtserkennung an, dann sind vorab zur Normierung eine größere Zahl von M typischen Aufnahmen auszuwerten. Die i-te Aufnahme mit n Zeilen und m Spalten interpretiert man als eine Folge von Pixelwerten, die in einem Vektor g entsprechend der Gleichung (3.5) mit der Dimension $N = n \cdot m$ abgelegt werden. Der Mittelwert aller Vektoren g, d.h. aller Aufnahmen, entspricht dann dem Bild einer mittleren Person und dient zur Beseitung des Mittelwertes bei den Einzelbildern:

$$g_i = (g_{11}, \ldots, g_{1m}, g_{21}, \ldots, g_{2m}, \ldots, g_{n1}, \ldots, g_{nm})^T \tag{3.5}$$

$$g_{Mittelwert} = \tfrac{1}{M} \sum_{i=1}^{M} g_i \quad \longrightarrow \quad d_i = g_i - g_{Mittelwert} \tag{3.6}$$

Die Vektoren d_i sind ein Maß für die Varianz der einzelnen Personenbilder in Relation zur mittleren Person und lassen sich als Spaltenvektoren in der Matrix D zusammenfassen. Die Matrix D besteht dabei aus aus N Zeilen und M Spalten. Durch die Multiplikation der Matrix D mit ihrer Transponierten ergibt sich eine symmetrische Kovarianzmatrix C mit der Ordnung $N = n \cdot m$:

$$C = DD^T = \begin{pmatrix} \vdots & \vdots & \vdots \\ d_1 & \ldots & d_M \\ \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} \ldots & d_1 & \ldots \\ \ldots & d_2 & \ldots \\ \vdots & \vdots & \vdots \\ \ldots & d_M & \ldots \end{pmatrix} = \begin{pmatrix} \sigma_{11}^2 & \ldots & \sigma_{1N}^2 \\ \vdots & \ddots & \vdots \\ \sigma_{N1}^2 & \ldots & \sigma_{NN}^2 \end{pmatrix} \tag{3.7}$$

Die C-Matrix nimmt nun die sehr hohe Ordnung N an. Bei einem Bild von z.B. 400x500 Pixeln ergibt N einen Wert von 200000. Davon nun die Eigenwerte zu berechnen ist problematisch. Die erforderlichen Eigenvektoren hätten ebenfalls 200000 Komponenten. Ein eleganter mathematischer Weg um die gesuchten Eigenvektoren zu erhalten besteht darin, die Reihenfolge der Multiplikationen der D-Matrizen zu vertauschen. Die entsprechende L-Matrix folgt dann aus:

$$L = D^T D = \begin{pmatrix} \ldots & d_1 & \ldots \\ \ldots & d_2 & \ldots \\ \vdots & \vdots & \vdots \\ \ldots & d_M & \ldots \end{pmatrix} \begin{pmatrix} \vdots & \vdots & \vdots \\ d_1 & \ldots & d_M \\ \vdots & \vdots & \vdots \end{pmatrix} = \begin{pmatrix} \sigma_{11}^2 & \ldots & \sigma_{1M}^2 \\ \vdots & \ddots & \vdots \\ \sigma_{M1}^2 & \ldots & \sigma_{MM}^2 \end{pmatrix} \tag{3.8}$$

Nun weist die L-Matrix nur noch einen Rang von M auf, d.h. bei z.B. 4 Bildern nur noch den Wert von 4. Auch die Eigenvektoren der L-Matrix bestehen dann nur aus M Komponenten. Bei z.B. vier Bildern mit jeweils 200x300 Pixeln besteht die quadratische L-Matrix nur vier Zeilen und vier Spalten und somit sind auch nur vier Eigenvektoren zu bestimmen. Der Rang der C-Matrix dagegen hätte 60.000 betragen und die Bestimmung der 60.000 Eigenvektoren schließt jede Echtzeit-Anwendung aus.

Allerdings muss nun noch ein Weg gefunden werden, um aus den Eigenvektoren der L-Matrix die Eigenvektoren der C-Matrix zu berechnen. Vereinfachend kommt hinzu, dass die Bildinformationen sich in den zahlenmäßig großen Eigenwerten abbilden, d.h. es genügt nur die ersten M Eigenvektoren zu bestimmen. Die folgenden Gleichungen zeigen, wie sich die Eigenwerte der C-Matrix aus den Eigenwerten der L-Matrix gewinnen lassen.

Mit den Eigenvektoren e der L-Matrix folgt:

$$Le_i = \lambda_i e_i \tag{3.9}$$
$$D^T De_i = \lambda_i e_i \tag{3.10}$$
$$DD^T De_i = \lambda_i De_i \tag{3.11}$$
$$CDe_i = \lambda_i De_i \tag{3.12}$$

Der Vergleich zwischen der Gleichung (3.9) und der Gleichung (3.12) zeigt, dass sich die Eigenvektoren v der C-Matrix aus den Eigenvektoren e der L-Matrix ergeben. Die Eigenvektoren e müssen dazu als Vektoren der Länge 1 vorliegen.

$$v_i = De_i \quad \longrightarrow \quad V = D\,[e_1, \cdots, e_M] \tag{3.13}$$

Die einzelnen ersten M Eigenvektoren v_i faßt man im Hinblick auf die Anwendung zur Bilderkennung in einer „Eigenbild-Matrix“ V zusammen. Diese V-Matrix besteht aus N Zeilen und M Spalten. Für den Anwendungsbereich der Gesichtserkennung bezeichnet man diese V-Matrix als den „Gesichtsraum“ und die entsprechenden Vektoren v als die „Eigengesichter“ . Die Gleichung (3.14) stellt die normierte V-Matrix und ihre Struktur symbolisch dar. Es handelt sich um eine rechteckige Matrix mit M Spalten und N Zeilen, wobei gilt:

$$N >> M \quad \longrightarrow \quad V = \begin{pmatrix} v_{11} & \cdots & v_{1M} \\ \vdots & \cdots & \vdots \\ v_{i1} & \cdots & v_{iM} \\ \vdots & \cdots & \vdots \\ v_{N1} & \cdots & v_{NM} \end{pmatrix} \tag{3.14}$$

Mittels der V-Matrix lassen sich die vorgegebenen M Bilder, die in Form der Vektordarstellung g_x vorliegen, mittelwertfrei in den Raum der Eigenbilder transformieren:

$$w_i = V^T(g_i - g_{Mittelwert}) = V^T d_i; \quad i = 1, \cdots, M \tag{3.15}$$

Anschließend bildet man die Differenzen r_{ij} zwischen allen Vektoren w_i, wobei jeder der Vektoren w_i aus M Komponenten besteht.

$$r_{i,j} = \sqrt{(w_i - w_j)^T(w_i - w_j)}; \quad i, j = 1, \cdots, M \tag{3.16}$$

$$r_{max} = maximum(r_{i,j}) \tag{3.17}$$

Der Grenzwert r_{max} dient zur Relativierung der Differenzen $r_{i,x}$ zwischen einem gespeicherten Bild i des Eigenraumes und einem gescannten und in den Eigenraum transformierten Bild x. Außerdem legt r_{max} fest, ob das unbekannte Bild als noch zum Bildraum gehörend interpretiert werden kann.

Die Transformation des gescannten Bildes in den Bildraum erfolgt mit Gleichung (3.18):

$$w_x = V^T(g_x - g_{Mittelwert}) = V^T d_x \tag{3.18}$$

Anschließend berechnet man den für den Bildvergleich erforderlichen Fehlerwert $r_{i,x}$ in einer relativen Form $\mu_{i,x}$, welche ein prozentuales Maß für den Grad der Übereinstimmung zwischen einem Bild i und dem Bild x darstellt.

$$r_{i,x} = \sqrt{(w_x - w_i)^T(w_x - w_i)}; \quad i = 1, \cdots, M \tag{3.19}$$

$$\mu_{i,x} = \left(1 - \frac{r_{i,x}}{r_{max}}\right) 100\%; \qquad i = 1, \cdots, M \tag{3.20}$$

Der relative Wert von $\mu_{i,x}$ läßt sich wie folgt interpretieren:

- Gilt für ein bestimmtes Bild i $\mu_{i,x} = 100\%$,
  dann stimmt das Bild i mit dem Bild x exakt überein.

- Gilt für ein bestimmtes Bild i $0 \leq \mu_{i,x} \leq 100\%$,
  dann gehört das Bild i zum Bildraum.

- Gilt für alle Bilder i $\mu_{i,x} \leq 0$,
  dann gehört das Bild x nicht zum Bildraum.

Die Möglichkeit die relativen M Fehlerwerte $\mu_{i,x}$ bestimmen zu können und der schnelle Algorithmus, bei dem die aufwändige Eigenvektor-Berechnung nur beim Programmstart anfällt, sowie die Option des Nachlernens von Bildern durch eine Wiederholung der Eigenvektor-Berechnung verdeutlichen die große Nützlichkeit der Eigenwert-Methode bei der Bilderkennung.

Beispiel 3.2:

Als anschauliches Beispiel einer automatischen Erkennung von Spielkarten eignen sich die vier „Buben“ eines Skatspieles, die gescannt und in Grauwert-Matrizen von 200x300 Pixel umgewandelt wurden. Die entsprechende D-Matrix besteht aus 60.000 Zeilen, aber nur aus vier Spalten. Die Berechnung der vier Eigenwerte der L-Matrix erfolgte mittels eines numerischen Berechnungsverfahrens, das im nächsten Abschnitt erläutert wird.

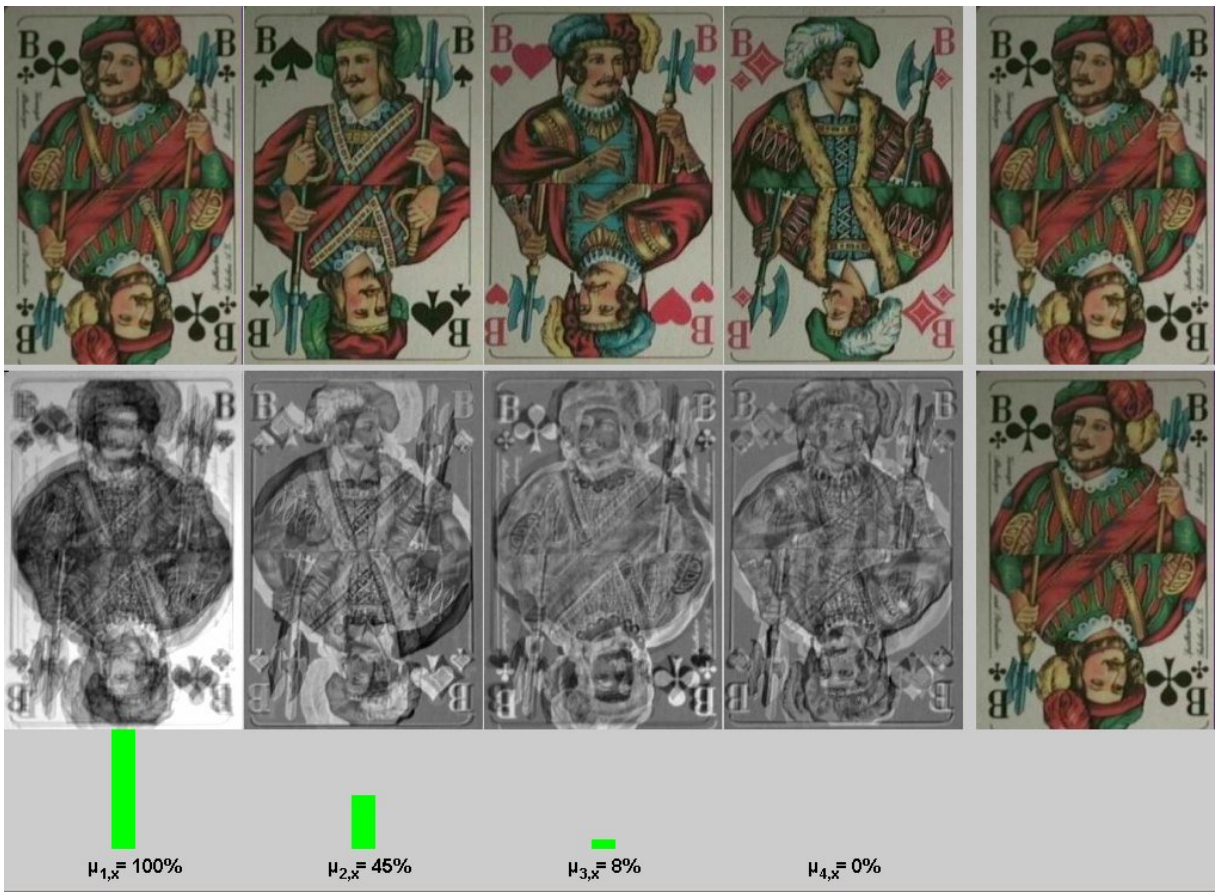


Abb.: 3.1: Erkennung einer identischen Spielkarte

Die ersten vier Bilder der oberen Reihe der Abb. 3.1 stellen jene Abbildungen dar, die als Basis für die Berechnung der L-Matrix dienen. In der unteren Reihe der Abb. 3.1 sind die Eigenbilder dargestellt, deren Grauwerte für die Anzeige auf dem Monitor mit einem offset versehen sind. Die rechts oben dargestellte Spielkarte wird eingescannt. Bei der Abb. 3.1 ist die eingescannte Spielkarte mit einer der gespeicherten vier Spielkarten identisch. Deshalb ergibt sich für den entsprechenden μ-Wert das erwartete Ergebnis von 100% Übereinstimmung. Die anderen μ-Werte liefern die entsprechend geringeren Werte. Einer der μ-Werte muss Null sein, denn in diesem Fall entspricht der Wert von r gleich dem maximalen Wert r_{max} und wie aus den Gleichungen (3.20) und (3.17) ersichtlich ist, folgt dann für die Spielkarte mit der maximalen Abweichung exakt der Wert Null. Das Bild in der unteren Reihe, das ganz rechts dargestellt und unterhalb des gescannten Bildes plaziert ist, zeigt das Ergebnis der Bilderkennung, das in diesem Sonderfall von μ=100% mit dem gespeicherten Bild des „Kreuz-Buben“ übereinstimmt.

In einem weiteren Testlauf scannt man nun wieder einen „Kreuz-Buben“ ein, der jedoch gegenüber dem gespeicherten Bild gedreht ist. Die Abb. 3.2 zeigt den entsprechenden Bildschirmausdruck. Obwohl das gespeicherte Bild vom gescannten Bild erheblich abweicht, stellt der Eigenwert-Algorithmus eine Übereinstimmung von 87% zum „Kreuz-Buben“ fest und weist gegenüber dem zweiten Bild eine Überstimmung von nur noch 57% auf. Damit demonstriert die Eigenwert-Methode ihre enorme Leistungsfähigkeit.

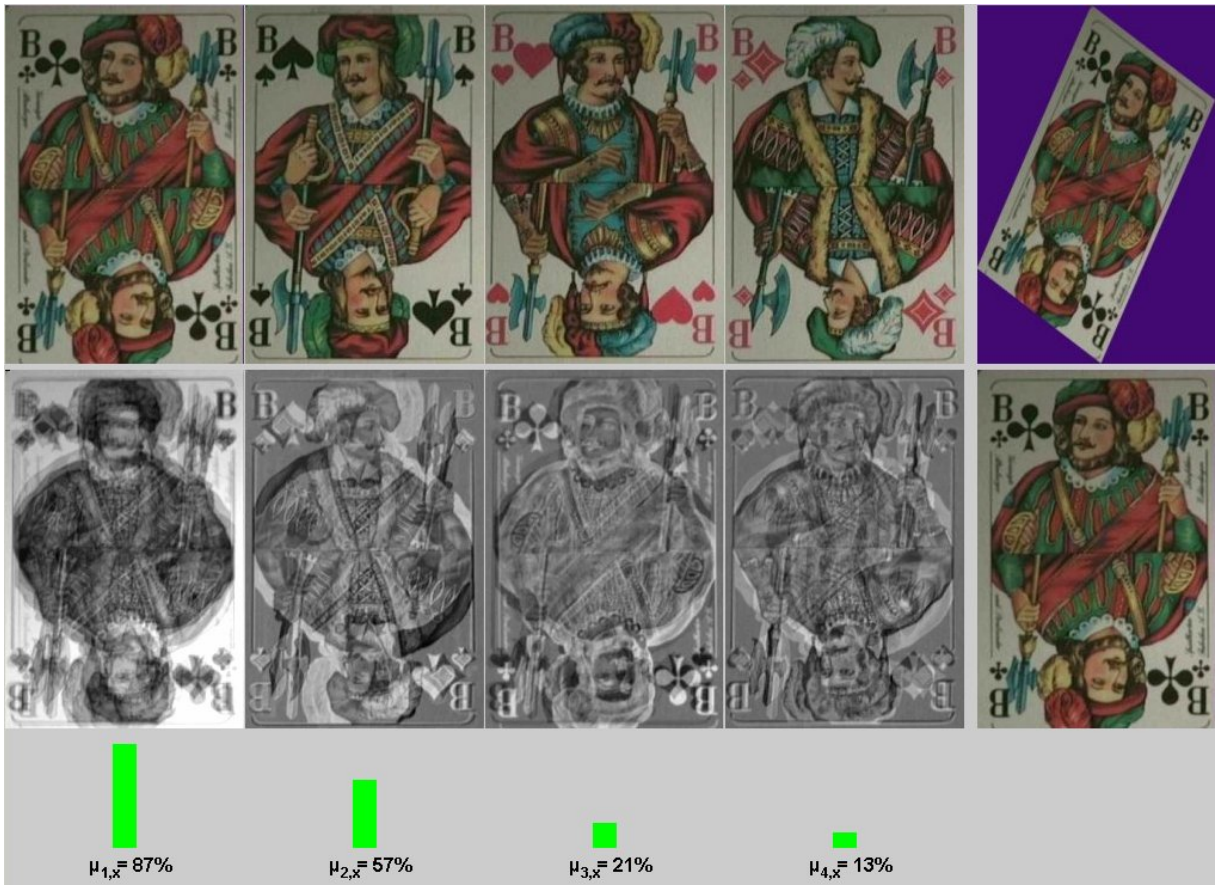


Abb.: 3.2: Ergebnis bei der Erkennung einer gedrehten Spielkarte

Der dritte Testlauf zeigt, welche Ergebnissse die Eigenwert-Methode liefert, wenn kein „Bube“, sondern das Bild der Mona Lisa gescannt wird. Alle μ-Werte wandern in diesem, in der Abb. 3.3 dargestellten Fall, in den negativen Bereich und damit folgt als Ergebnis, dass die gescannte Aufnahme keiner der in der L-Matrix abgebildeten Spielkarten entspricht.

Insgesamt demonstriert das Beispiel 3.1 sehr anschaulich die Stärken der Eigenwert-Methode. Als weiterer Vorteil darf nicht unerwähnt bleiben, dass es sich bei dieser Methode um einen

sehr schnellen Algorithmus handelt. Die Algorithmen, die nur beim Programmstart aufgerufen werden, laufen auf einem Standard-PC innerhalb von weniger als 4 Sekunden ab. Der eigentliche Erkennungsvorgang liegt für diese 200x300 Pixel großen Bilder im Bereich von Millisekunden.

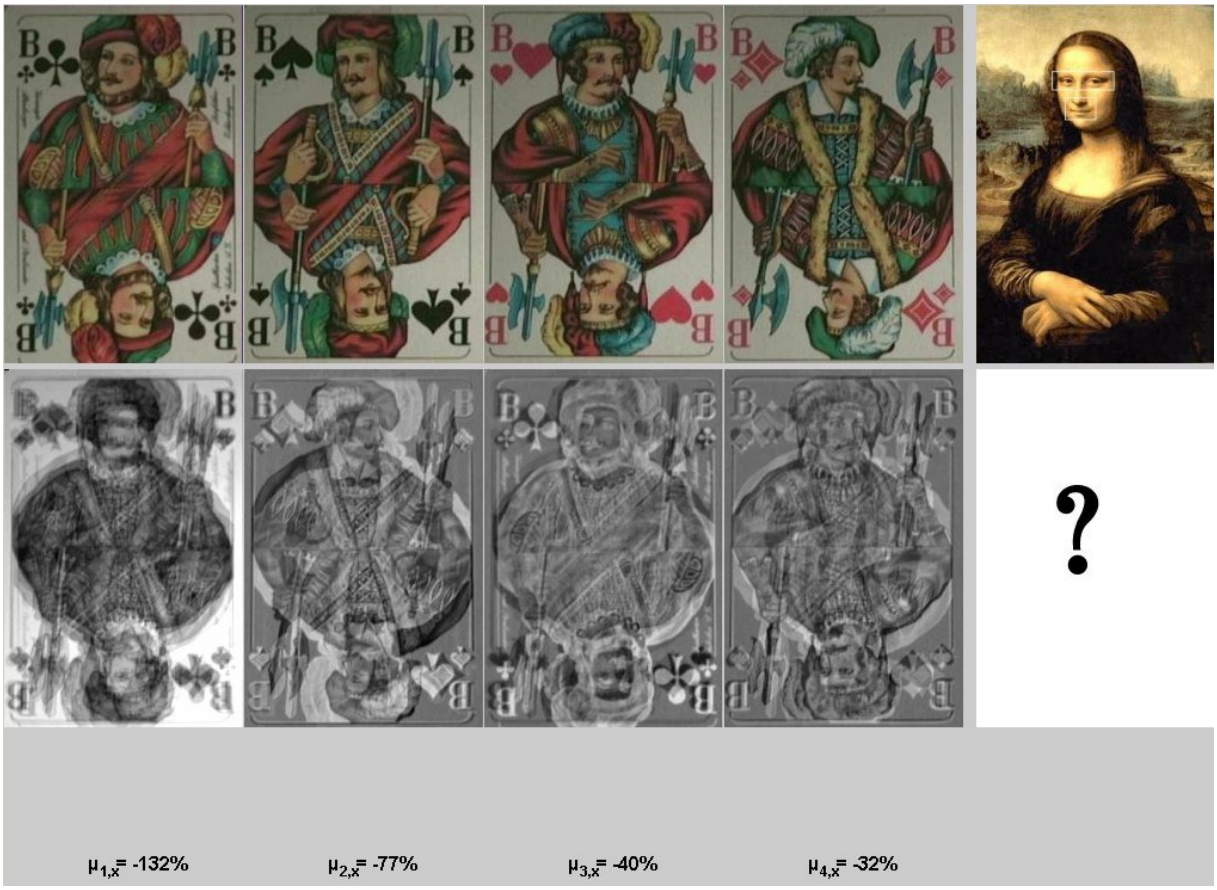


Abb.: 3.3: Scannen eines Bildes, das keine Spielkarte ist

Das in der Abb. 1.4 visualisierte Ergebnis verdeutlicht die wichtigste Eigenschaft der Eigenwert-Methode für die Bilderkennung:

> Die Eigenwert-Methode besitzt die Fähigkeit zu erkennen, ob ein graphisches Objekt überhaupt bzw. mit welchem positiven Prozentsatz zur Gruppe der gespeicherten Bilder gehört .

Für das Anwendungsgebiet der Gesichtserkennung ist diese Eigenschaft von größter Bedeutung, denn sonst bestünde die Gefahr, dass die Erkennungsalgorithmen graphische Objekte analysieren, die gar keine Gesichter sind und dann falsche unverantwortliche Ergebnisse die Folge wären.

Als sehr hilfreich erweist sich das relative Maß μ zur Bewertung der Zuverlässigkeit einer automatischen Erkennung. Positive, jedoch geringe Werte für μ verursachen unsichere Ergebnisse des Erkennungsverfahrens. So besteht dann z.B. die Möglichkeit, durch eine Nachführung der Kamera den Wert μ zu maximieren. Die geringe Rechenzeit, die für die Transformation der gescannten Bilder in den Eigenraum erforderlich ist, ermöglicht ein ausreichend schnelles Nachführen der Überwachungskamera.

3.3 Numerische Eigenvektor-Berechnung

Wendet man die Eigenwert-Methode zur Bilderkennung an, dann erfolgt die Berechnung der Eigenvektoren mittels eines numerischen, iterativen Verfahrens. Die relativ komplizierte Berechnung der Eigenwerte und der Eigenvektoren vereinfacht sich in diesem Anwendungsfall zum Glück erheblich, denn die Ausgangsmatrizen sind grundsätzlich symmetrisch und somit existieren gemäß der Theorie der Eigenwerte nur reelle Eigenwerte.

Diese Eigenwerte der L-Matrix beginnen bei einem maximalen Wert und nehmen dann laufend ab. Die wichtigen Bildinformationen beschreiben die großen Eigenwerte und daher lassen sich alle betragsmäßig kleinen Eigenwerte vernachlässigen. In aller Regel ist es ausreichend, nur eine Anzahl von Eigenwerten zu berücksichtigen, die der Anzahl der Bilder entspricht und damit mit dem Rang M der L-Matrix übereinstimmt.

Unter diesen Voraussetzungen hat das numerische Verfahren zur Eigenvektor-Berechnung nach „von Mises“ eine große Bedeutung erlangt. Mit diesem Verfahren ermittelt man iterativ grundsätzlich den größten Eigenwert zusammen mit dem entsprechenden Eigenvektor. Mittels der Gleichung

$$L1 = L - \lambda_{max}\ (x_{max}\ x_{max}^T) \tag{3.21}$$

entsteht eine neue symmetrische Matrix L1, die den Eigenwert λ_{max} nicht mehr enthält bzw. dessen Wert nun Null beträgt. Anschließend beginnt das Verfahren wieder neu und berechnet den nächst größten Eigenwert.

Die Grundidee des Verfahrens besteht darin, dass sich ein beliebiger, mit der Matrix L transformierter Vektor z, nach den Eigenvektoren der L-Matrix entwickeln läßt:

$$w_i = Lz_{i-1} = c_1\lambda_1^i x_1 + \cdots + c_M\lambda_M^i x_M \tag{3.22}$$

$$z_{i-1} = w_i \quad \text{für} \quad i = 2, 3, \cdots, imax \tag{3.23}$$

Durch das mehrfache Multiplizieren überwiegt nach einer ausreichenden Anzahl von $imax$ Schritten der größte Eigenwert von z.B. λ_1:

$$w_{imax} = c_1\ \lambda_1^{imax}\ x_1 \tag{3.24}$$

Das Verhältnis zweier aufeinander folgender w-Vektoren konvergiert gegen den größten Eigenwert. Die Vektoren w nehmen den Wert des entsprechenden Eigenvektors an. Durch einen Normierungsfaktor k_i mit dem Betrag des w-Vektors entsteht ein einfacher und numerisch stabiler Algorithmus:

$$w_i = Lz_{i-1} \tag{3.25}$$

$$k_i = \sqrt{w_i^T\ w_i} \tag{3.26}$$

$$z_i = \frac{1}{k_i}\ w_i \quad i = 1, 2, 3, \cdots, imax \tag{3.27}$$

Diesen Algorithmus wiederholt man bis zur Konvergenz und erhält dann an der Stelle $i = imax$ den gesuchten Eigenwert bzw. den für die V-Matrix erforderlichen Eigenvektor:

$$e_i = z_{imax} \tag{3.28}$$

$$\lambda = k_{imax} \tag{3.29}$$

Beispiel 3.3:

Das Beispiel 3.2 veranschaulicht an Hand der in der Abb. 3.4 dargestellten Bilder den grundsätzlichen algorithmischen Verlauf der Eigenwert-Methode. Das linke und mittlere Bild der Abb. 3.4 repräsentiert die beiden gespeicherten Bilder und das rechte Bild der Abb. 3.4 entspricht einer gescannten Aufnahme.

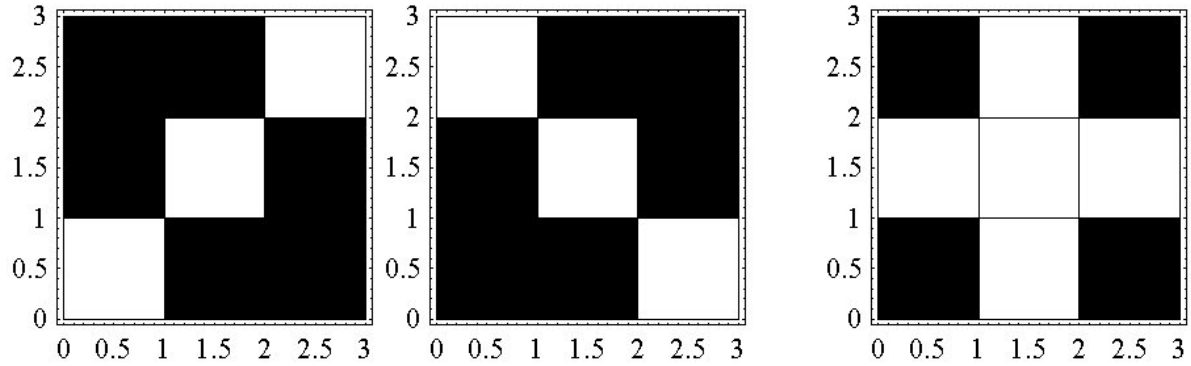


Abb.: 3.4: Beispielbilder zur Demonstration der Eigenwert-Methode

Die Zahlenwerte 0 bzw. 2 entsprechen einem weißem bzw. einem gesetzten schwarzen Pixel. Der Wert 2 soll die nun folgenden manuellen numerischen Berechnungen überschaubar halten. Für die beiden Bildvektoren g_1, g_2 folgen mit dem Mittelwert-Vektor d_M die Differenz-Vektoren d_1, d_2 und der unbekannte Differenz-Vektor d_x:

$$g_1 = \begin{pmatrix} 2 \\ 2 \\ 0 \\ 2 \\ 0 \\ 2 \\ 0 \\ 2 \\ 2 \end{pmatrix} \quad g_2 = \begin{pmatrix} 0 \\ 2 \\ 2 \\ 2 \\ 0 \\ 2 \\ 2 \\ 2 \\ 0 \end{pmatrix} \quad g_M = \begin{pmatrix} 1 \\ 2 \\ 1 \\ 2 \\ 0 \\ 2 \\ 1 \\ 2 \\ 1 \end{pmatrix} \quad d_1 = \begin{pmatrix} 1 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 1 \end{pmatrix} \quad d_2 = \begin{pmatrix} -1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ -1 \end{pmatrix} \quad g_x = \begin{pmatrix} 2 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 2 \end{pmatrix} \quad d_x = \begin{pmatrix} 1 \\ -2 \\ 1 \\ -2 \\ 0 \\ -2 \\ 1 \\ -2 \\ 1 \end{pmatrix}$$

$$L = D^T D = \begin{pmatrix} d_1 & d_2 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \begin{pmatrix} 1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 1 \\ -1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 0 & 0 \\ -1 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -1 & 1 \\ 0 & 0 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} 4 & -4 \\ -4 & 4 \end{pmatrix}$$

Die Berechnung der beiden Eigenwerte der L-Matrix basiert auf dem „von Mises“-Verfahren. Als Startwert für die Iteration wählt man einen beliebigen Wert, z.B.

$$z_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

1.) Berechnung des ersten Eigenvektors e_1

$$w_1 = Lz_0 = \begin{pmatrix} 4 & -4 \\ -4 & 4 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 4 \\ -4 \end{pmatrix} \quad \rightarrow \quad z_1 = \frac{1}{\sqrt{4^2+4^2}} \begin{pmatrix} 4 \\ -4 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

$$w_2 = Lz_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 4 & -4 \\ -4 & 4 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} \frac{8}{\sqrt{2}} \\ -\frac{8}{\sqrt{2}} \end{pmatrix} \quad \rightarrow \quad z_2 = \frac{1}{\sqrt{32+32}} \begin{pmatrix} \frac{8}{\sqrt{2}} \\ -\frac{8}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

Damit liegt der erste Eigenwert e_1 bereits nach drei Iterationen fest.

$$\lambda_1 = \sqrt{32+32} = 8 \quad \text{bzw.} \quad e_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

Vor der Berechnung des zweiten Eigenwertes führt man die Reduktion der L-Matrix gemäß der Gleichung (3.21) durch. Dann startet das Verfahren mit der Matrix L1 wieder neu:

$$\begin{aligned} L1 = L - \lambda_1 \, (e_1 \, e_1^T) &= \begin{pmatrix} 4 & -4 \\ -4 & 4 \end{pmatrix} - 8 \, \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} \, \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \end{pmatrix} \\ &= \begin{pmatrix} 4 & -4 \\ -4 & 4 \end{pmatrix} - 4 \, \begin{pmatrix} 1 \\ -1 \end{pmatrix} \, \begin{pmatrix} 1 & -1 \end{pmatrix} \\ &= \begin{pmatrix} 4 & -4 \\ -4 & 4 \end{pmatrix} - 4 \, \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 4 & -4 \\ -4 & 4 \end{pmatrix} - \begin{pmatrix} 4 & -4 \\ -4 & 4 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \end{aligned}$$

2.) Berechnung des 2. Eigenvektors e_2

Nach der Subtraktion des ersten Eigenwertes nahm die L1-Matrix den Wert 0 an. Deshalb entfällt in diesem Beispiel nun die Wiederholung des interativen Verfahrens und man erhält die folgenden Ergebnisse:

$$\begin{pmatrix} 4-\lambda_2 & -4 \\ -4 & -4-\lambda_2 \end{pmatrix} e_2 = 0 \quad \text{mit} \quad \lambda_2 = 0 \quad \longrightarrow \quad \begin{pmatrix} 4 & -4 \\ -4 & 4 \end{pmatrix} \begin{pmatrix} e_{2x} \\ e_{2y} \end{pmatrix} = 0$$

$$I : 4 \, e_{2x} - 4 \, e_{2y} = 0 \qquad II : -4 \, e_{2x} + 4 \, e_{2y} = 0$$

Aus den linear abhängigen Gleichungen ergibt sich der Eigenvektor e_2.

$$\frac{e_{2x}}{e_{2y}} = \frac{1}{1} \quad \longrightarrow \quad e_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Die beiden Eigenvektoren e_1 und e_2 sind nun mit der D-Matrix in die Eigenwerte v_1 und v_2 der C-Matrix zu transformieren.

$$e_1 = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix} \longrightarrow v_1 = De_1 = \begin{pmatrix}1&-1\\0&0\\-1&1\\0&0\\0&0\\0&0\\-1&1\\0&0\\1&-1\end{pmatrix} \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}2\\0\\-2\\0\\0\\0\\-2\\0\\2\end{pmatrix}$$

$$e2 = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} \longrightarrow v_2 = De_2 = \begin{pmatrix}1&-1\\0&0\\-1&1\\0&0\\0&0\\0&0\\-1&1\\0&0\\1&-1\end{pmatrix} \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\end{pmatrix} = \begin{pmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\end{pmatrix}$$

Durch die Zusammenfassung der Eigenvektoren v_1 und v_2 in einer Matrix entsteht die Transformationsmatrix V:

$$V = (v_1, v_2) \longrightarrow V = \begin{pmatrix}\sqrt{2}&0\\0&0\\-\sqrt{2}&0\\0&0\\0&0\\0&0\\-\sqrt{2}&0\\0&0\\\sqrt{2}&0\end{pmatrix}$$

Die Transformations-Matrix V enthält in den Spalten die zu den jeweiligen Eigenwerten gehörenden Eigenbilder. Alle Bildvektoren d sind mit dieser V-Matrix in den Eigenraum zu transformieren, wobei die V-Matrix in der transponierten Form auftritt:

$$w_i = V^T d_i \; ; \quad i, 1, \cdots, M \qquad \text{bzw.} \quad w_x = V^T d_x \tag{3.30}$$

Die Berechnung der Eigenbilder w_i erfolgt beim Start des Programms. Während der eigentlichen Bilderkennung ist nur noch die rechte Seite der Gleichung (3.30) auszuwerten. Die vorab einmal berechnete V-Matrix und der entsprechende Mittelwert-Vektor sind daher abzuspeichern. Damit erhält man einen schnellen Algorithmus.

$$w_1 = V^T d_1 = \begin{pmatrix} \sqrt{2} & 0 & -\sqrt{2} & 0 & 0 & 0 & -\sqrt{2} & 0 & \sqrt{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 4\sqrt{2} \\ 0 \end{pmatrix} = \begin{pmatrix} \sqrt{32} \\ 0 \end{pmatrix}$$

$$w_2 = V^T d_2 = \begin{pmatrix} \sqrt{2} & 0 & -\sqrt{2} & 0 & 0 & 0 & -\sqrt{2} & 0 & \sqrt{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} -1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} -4\sqrt{2} \\ 0 \end{pmatrix} = \begin{pmatrix} -\sqrt{32} \\ 0 \end{pmatrix}$$

$$w_x = V^T d_x = \begin{pmatrix} \sqrt{2} & 0 & -\sqrt{2} & 0 & 0 & 0 & -\sqrt{2} & 0 & \sqrt{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ -2 \\ 1 \\ -2 \\ 0 \\ -2 \\ 1 \\ -2 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Im Eigenraum, der die Vektoren w_1, w_2 und w_x enthält, erfolgt schließlich der eigentliche Bildvergleich. Der Grenzwert ϵ ergibt sich aus der Betragsdifferenz der Vektoren w_1, w_2 und als d_x wird jenes Bild erkannt, das mit den Bildern w_1 und w_2 die größte Überstimmung aufweist.

$$r_{1,2}^2 = (w_1 - w_2)^T (w_1 - w_2) = \left(\sqrt{32} - (-\sqrt{32}) \quad 0 - 0)\right) \begin{pmatrix} \sqrt{128} \\ 0 \end{pmatrix} = \sqrt{128}$$

$$r_{1,x}^2 = (w_1 - w_x)^T (w_1 - w_x) = \left(\sqrt{32} - 0 \quad 0 - 0)\right) \begin{pmatrix} \sqrt{32} \\ 0 \end{pmatrix} = \sqrt{32}$$

$$r_{2,x}^2 = (w_2 - w_x)^T (w_2 - w_x) = \left(-\sqrt{32} - 0 \quad 0 - 0)\right) \begin{pmatrix} -\sqrt{32} \\ 0 \end{pmatrix} = \sqrt{32}$$

Als nächstes folgt nun die Bestimmung des r_{max}-Wertes. Dazu sind die Vektordifferenzen zwischen allen einzelnen r-Werten zu berechnen und der Betrag zu bilden. Bei z.B. 6 Bildern wären dies die Kombinationen:

$$\begin{array}{lllll} r_{1,2} & r_{1,3} & r_{1,4} & r_{1,5} & r_{1,6} \\ r_{2,3} & r_{2,4} & r_{2,5} & r_{1,6} & \\ r_{3,4} & r_{3,5} & r_{3,6} & & \\ r_{4,5} & r_{4,6} & & & \\ r_{5,6} & & & & \end{array}$$

Da in dem vorliegenden einfachen Beispiel nur zwei Bilder den Eigenraum aufspannen, entspricht der Wert von $r_{1,2}$ direkt dem Wert r_{max}:

$$r_{max} = \sqrt{r_{1,2}}$$

$$\mu_{1,x} = (1 - \frac{\sqrt{r_{1,x}}}{r_{max}})\ 100\% = (1 - \frac{\sqrt{32}}{\sqrt{128}})\ 100\% = 50\% \quad \rightarrow \quad 0 \leq \mu_{1,x} \leq 100\%$$

$$\mu_{2,x} = (1 - \frac{\sqrt{r_{2,x}}}{r_{max}})\ 100\% = (1 - \frac{\sqrt{32}}{\sqrt{128}})\ 100\% = 50\% \quad \rightarrow \quad 0 \leq \mu_{2,x} \leq 100\%$$

Damit konnte das gestörte Bild x als zum Bildraum gehörend erkannt werden. Die Übereinstimmung mit den vorhandenen Bildern beträgt allerdings nur 50%.

Bei diesem einfachen Zahlenbeispiel treten die wesentlichen Eigenschaften der Eigenwert-Methode zur Bilderkennung deutlich hervor:

- Es handelt sich um ein schnelles Verfahren der Bilderkennung, denn die Berechnung der Eigenwerte erfolgt nur einmal und zwar beim Start des Programmes .
- Die Ermittlung der prozentualen $\mu_{i,x}$-Werte als Maß der Übereinstimmung zwischen den aktuell gescannnten und den gespeicherten Bildern läßt sich vielseitig anwenden, wie z.B. bei der automatischen Aufnahme sicher erkannter Bilder in die D-Matrix.
- Bei einer Veränderung der Bilddatenbank müssen die V- und W-Matrix neu berechnet werden.
- Das Nachlernen bzw. das Vergessen von Bildern erfolgt durch ein update der Eigenvektoren.
- Die Helligkeitswerte der Pixelwerte müssen sorgfältig normiert werden, damit die Bilder vergleichbare r-Werte liefern.

Kapitel 4

Neuronale Netze

4.1 Biologische Grundlagen neuronaler Netze

Bei der Entwicklung und Anwendung künstlicher neuronaler Netze geht es darum, biologische Gehirnfunktionen in Form von Computerprogrammen zu simulieren, um damit die enorme Leistungsfähigkeit der Biologie im Bereich der Schrift-, Sprach- und Bilderkennung für technische Anwendungen erschließen zu können. Die Tatsache, dass die aktuelle Computertechnologie bei den genannten kognitiven Anwendungen im Vergleich zu den menschlichen Fähigkeiten so kläglich versagt, war Hauptmotor für die Entwicklung künstlicher neuronaler Netze.

Grundsätzlich hat die Menschheit bezüglicher technischer Entwicklungen immer schon versucht, die Natur durch technische Geräte nachzubilden und wenn möglich dabei auch noch zu übertreffen. So führte z.B. das Studium des Vogelfluges zu den ersten Fluggeräten. Hundert Jahre später übertrifft ein Langstrecken-Airbus die Flugleistung der Vögel bei weitem. Allerdings gelingt es noch nicht, die Manövrierfähigkeit von Vögeln mit kleinen technischen Fluggeräten, wie z.B. mit kleinen militärischen Beobachtungshubschraubern, auch nur annähernd zu erreichen.

Der Vogelflug-Vergleich lässt sich auf die technische Nachbildung biologischer neuronaler Netze übertragen. Zeitlich betrachtet steht die Menschheit bei der Entwicklung neuronaler Netze am Anfang einer fantastischen Entwicklung, an dessen vorläufigem absehbaren Ende eine denkende, sprechende und verantwortlich handelnde Maschine stehen könnte. In vielen Zukunftsfilmen ist diese Vision bereits Realität, man denke nur an den sympathischen „LtCommander Data“ aus den populären Startrek Sfi-Serien:

Abb.: 4.1: Zukunftsvision: Der Schauspieler Brent Spinner als „LtCommander Data“

Doch wie sieht so ein biologisches Gehirn aus und besser noch, wie funktioniert es ? Betrachtet man dazu das am besten erforschte Tier der Welt, den unscheinbaren Fadenwurm (siehe Abb. 4.2) mit der lateinischen Bezeichnung „Caenorhabditis elegans“, der sich in Blumentöpfen und im Kompost heimisch fühlt und eine eigene Homepage [25] besitzt

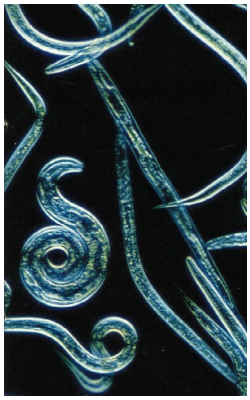


Abb.: 4.2: Fadenwürmer (Länge ca. 1mmm), nach [25]

dann lässt sich eine Gehirnstruktur erkennen, die ca. 1 mmm^3 Volumen aufweist und wie in der Abb. 4.3 dargestellt aussieht. Das ganze erinnert an ein zufällig geknüpftes Fischernetz, das die verschiedenen Typen von Neuronen verbindet.

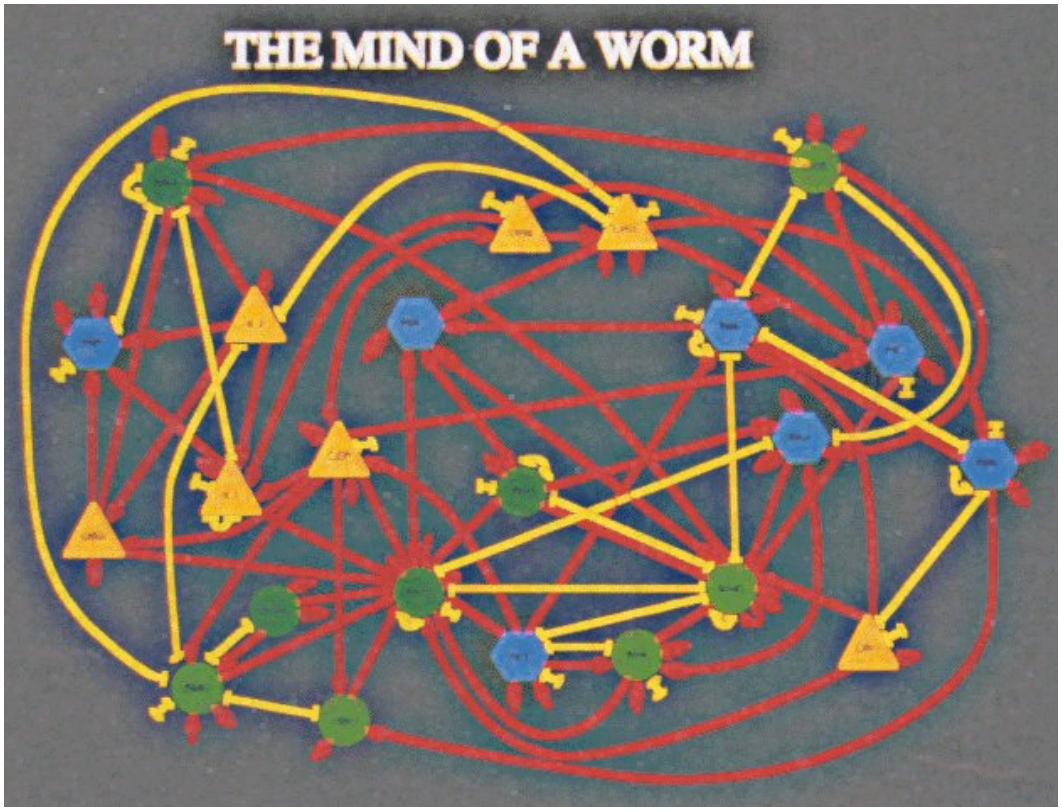


Abb.: 4.3: Die 302 Neuronen des Fadenwurms, nach [25]

Beim menschlichen Gehirn sieht ein mikroskopischer Schnitt ähnlich aus, allerdings erreicht die Anzahl der Neuronen den Milliarden-Bereich.

Abb.: 4.4: Schnitt durch ein Gehirn, nach [26]

Die Abb. 4.4 stellt einen Ausschnitt aus dem Nerven-Netzwerk des visuellen Cortex eines Kindes dar. Es sind Nervenzellen in Form von knotenartigen Verdickungen und deren Vernetzung mit unterschiedlich dicken Nervenleitungen sichtbar.

Die aktuellsten Gehirnscans, wie z.B. die Abb. 4.5, zeigen hoch aufgelöste winzige Ausschnitte der komplexen Gehirnstrukur. Bei den Neuronen lässt sich zwischen Eingängen und Ausgängen deutlich unterscheiden, wobei die Ausgänge den größeren Querschnitt aufweisen.

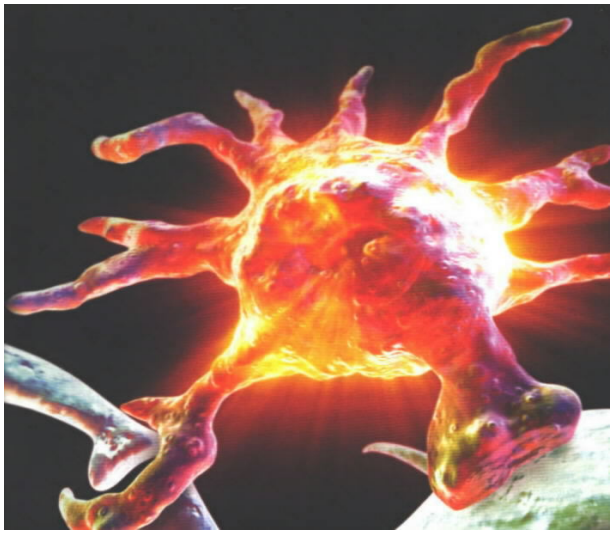


Abb.: 4.5: Visualisierung eines Neurons, nach [15]

In der Abb. 4.6 erkennt man, wie die Ausgänge der Knoten („Axon“) über die „Dendriten“ wieder zu den Eingängen anderer Neuronen führen. Der „synaptische Spalt“ bildet den Übergang von der Eingangsleitung in den Kern des Neurons.

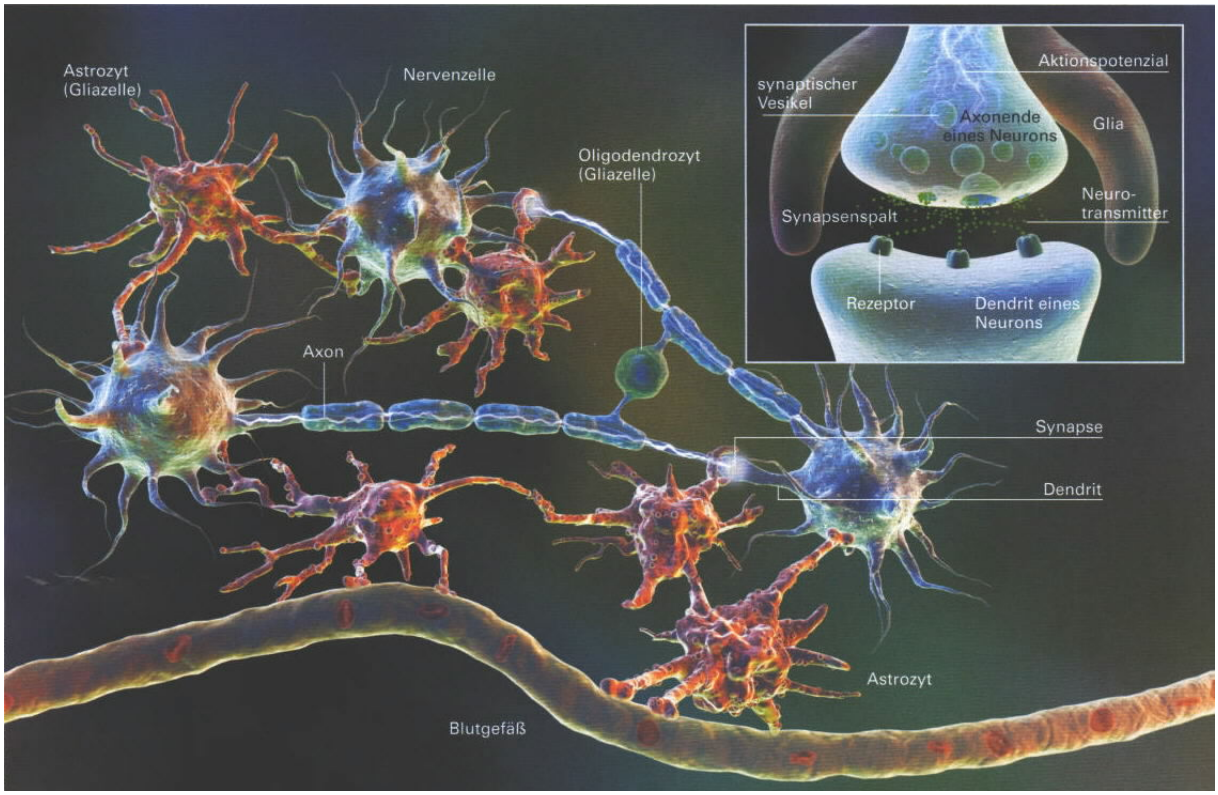


Abb.: 4.6: Darstellung eines Neuronen-Netzes nach [15]

Betrachtet man Neuronen im zeitlichen Verlauf ihrer Aktivitäten, dann lassen sich dynamische elektrische Potentialverläufe auf den Nervenbahnen nachweisen. Summieren sich die Eingangspegel eines einzelnen Neurons zu einem bestimmten Schwellwert auf, dann „feuert“ das Neuron, d.h. das Ruhepotential auf dem Axon geht kurzzeitig in einen pulsierenden positiven Potentialverlauf über und fällt dann wieder in den Ruhezustand mit einem negativen Potential zurück. Der Ruhepegel liegt bei ca. 80 mV und im pulsierenden Zustand erreicht das Potential auf der Axon-Leitung ca. +40 mV. Der positive Ausgangsimpuls erreicht als Eingangssignal dann andere Knoten und löst dort vielleicht wieder einen Impuls aus, d.h. diese Struktur weist ein dynamisches Verhalten auf.

Ob ein Neuron feuert oder nicht, hängt von den Eigenschaften des synaptischen Spalts ab. Je nach Zustand dieses Spaltes, gelangen die Eingangsimpulse direkt in das Neuron oder werden „gehemmt“. Gefühle lösen eine ständige Variation in den Parametern der synaptischen Verbindungen aus. Bemerkt ein Lebewesen positive Gefühle bei bestimmten Handlungen, dann werden die synaptischen Verbindungen enthemmt bzw. bei negativen Gefühlen gehemmt. Das biologische Lebewesen verändert auf der Basis von Erfahrungen laufend die Parameter des synaptischen Spaltes und adaptiert sich so an wechselnde Umgebungsbedingungen. Eine weitere Zellenart, die „Glia-Zellen“ umgeben den synaptischen Spalt und ermöglichen unabhängig von den Axon-Verbindungen eine Kommunikation zwischen Zellen.

Die Erkenntnisse über das biologische Gehirn wurden durch faszinierende Forschungsarbeiten in den 60-er Jahren gewonnen. Nun war es tatsächlich nur noch ein kleiner Schritt zur Realisierung des ersten künstlichen Neurons. Die mathematische Struktur zur Beschreibung der Neuronenfunktionen wirkt auf den ersten Blick trivial: Es ist die gewichtete Addition von Eingangssignalen („Netto-Input“) und das Ausgeben eines Ausgangssignals, falls die Summe einen Grenzwert überschreitet. Die Schaltfunktion nennt man Sigmoidfunktion.

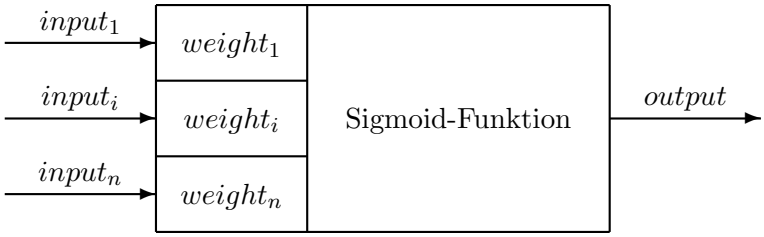


Abb.: 4.7: Mathematisches Modell eines Neurons

Die Gleichung (4.1) beschreibt den mathematischen Zusammenhang dieses künstlichen Neurons. Der Parameter Θ entspricht einem offset und dient zur Zentrierung des Netto-Inputs bezüglich beliebiger Sigmoidfunktionen. Durch eine entsprechende Skalierung der Eingangs- und Ausgangssignale auf den Zahlenbereich ± 1 entfällt dieser offset Θ.

$$output = \text{Sigmoidfunktion}\left(\sum_{i=1}^{i=n}(weight_i \cdot input_i) - \Theta\right) \tag{4.1}$$

Eine harte Schaltfunktion („hard limiter“) erweist sich in der praktischen Handhabung wegen der extremen Nichtlinearität als weniger geeignet. Man wählt daher Übergänge mit steiler Kennlinie, ähnlich den Transistor-Schaltfunktionen. Die Funktion des tanh bietet im Gegensatz zur klassischen Sigmoid-Funktion einen negativen Wertebereich von $\pm$ 1. Durch entsprechende Konstanten α vor der Variablen x läßt sich die Steilheit der

Kennlinien einstellen. Im Hinblick auf die Einsparung von Rechenzeit ist es möglich, mit wenigen Koeffizienten den stetigen Verlauf der tanh-Funktion näherungsweise in eine Potenzreihe rascher Konvergenz zu entwickeln.

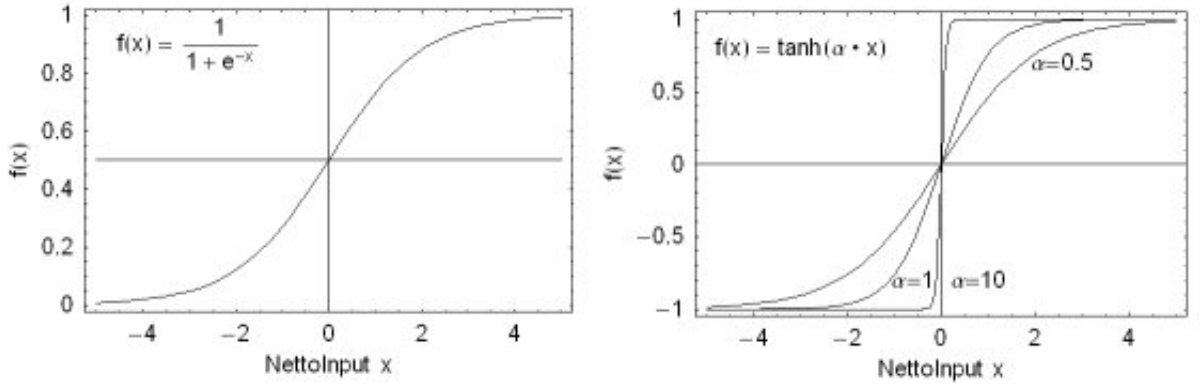


Abb.: 4.8: Beispiele für Sigmoid-Funktionen

Grundsätzlich lassen sich die künstlichen Neuronen so zu einem Netz verknüpfen, daß jedes Neuron mit jedem anderen vollständig verbunden ist. Der Ausgang eines einzelnen Neurons gelangt dann nicht nur auf den eigenen Eingang zurück, sondern auch auf die Eingänge aller anderen Neuronen. Diese Methode erweist sich jedoch im Hinblick auf praktische Anwendungen als sehr problematisch. Ein vollständig vorwärts und rückwärts gekoppeltes Netz zeigt chaotisches oder oszillierendes Verhalten und läßt sich nur mit einer geringen Anzahl von Neuronen überhaupt beherrschen. Tatsächlich stecken in dieser Topologie jedoch die zentralen Eigenschaften einer künstlichen Intelligenz. Um sich diesem anspruchsvollen Ziel zu nähern und um nützliche Anwendungen realisieren zu können, ist es allerdings vernünftig, zunächst nicht von der vollständigen Vernetzung auszugehen, sondern mit einfachen symmetrischen und berechenbaren Topologien zu beginnen.

Das Perceptron-Modell ist ein vorwärtsgerichtes Modell mit einer Eingangsschicht (input layer), einer Ausgangsschicht(output layer) und einer oder mehreren Zwischenschichten, die als Hidden Layer bezeichnet werden. Mathematisch betrachtet ist dieses Netz durch wenige Parameter bestimmt:

N: Anzahl der Neuronen in der Eingangsschicht

n: Anzahl der Signalleitungen je Eingangsneuron

K: Anzahl der Neuronen in der Zwischenschicht

L: Anzahl der Neuronen in der Ausgangsschicht

Bei vorgebenen Parametern N, n, K und L liegt auch die Anzahl der zu bestimmenden Gewichte fest. Für das in Abb. 4.9 dargestellte Netz ergeben sich somit 23 Gewichte:

$$\begin{aligned} N &= 3 \text{ und } M = 3 \quad \longrightarrow \quad 3 \cdot 3 = 9 \text{ Gewichte} \\ K &= 2 \quad \longrightarrow \quad 2 \cdot 3 = 6 \text{ Gewichte} \\ L &= 4 \quad \longrightarrow \quad 4 \cdot 2 = 8 \text{ Gewichte} \end{aligned}$$

Als mögliche Anwendung eines solches Netzes könnte man drei binäre Signale mit einer Wertigkeit von 0,1,2,3 an den Eingang legen. Die Ausgangsleitungen (0/1) entsprächen

dann vier Handlungsanweisungen. Für die Anwendung des Perceptron Netzes bei einer automatischen Maschinensteuerung entsprechen die drei Eingangssignale den Sensordaten und die Ausgänge steuern dann in optimaler Weise die Maschine. Bei der Bilderkennung reichen drei Signalleitungen am Eingang nicht mehr aus. Mit M=64 wäre nur ein 8x8-Pixelmuster möglich.

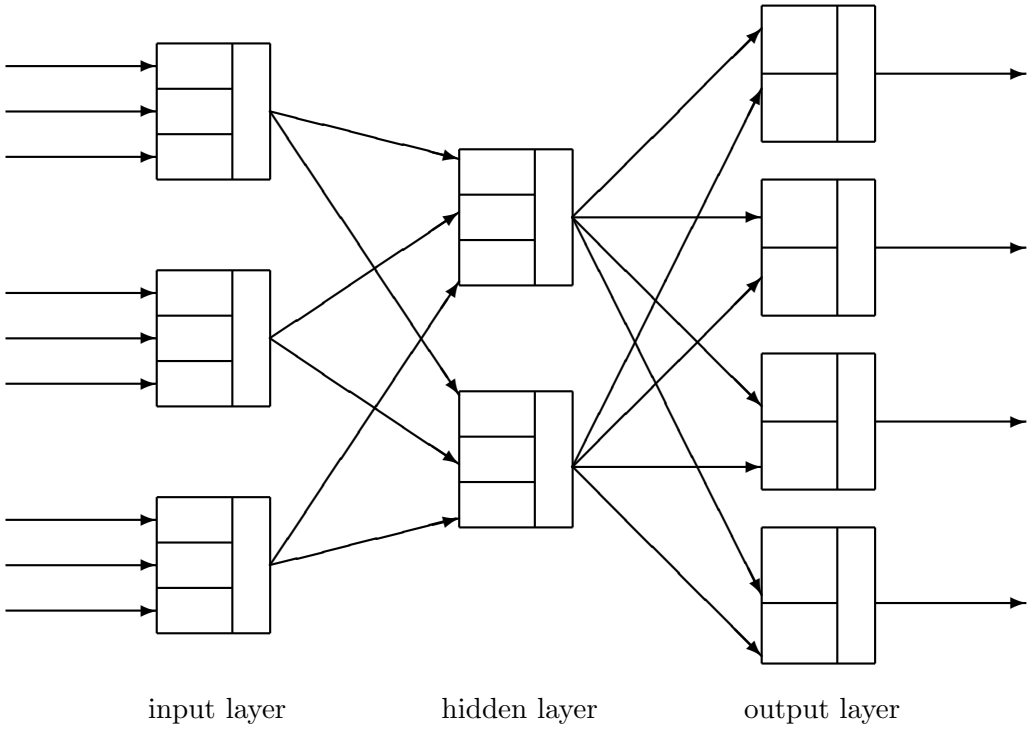


Abb.: 4.9: Topologie eines Perceptron Netzes

Der Rechenaufwand für die Auswertung des Netzes ist prinzipiell vernachlässigbar. Der eigentliche Aufwand steckt in der Bestimmung der Gewichte. Dies hängt mit dem Lernvorgang zusammen. Wie bekannt, ist ein Gedicht schnell aufgesagt, aber es dauert lange, um es auswendig zu lernen. Der Musiker übt ein Leben lang, d.h. er trainiert die entsprechenden Gewichte (Synapsen) im Gehirn. Das scheinbar mühelose Präsentieren von Kenntnissen und Fertigkeiten ist immer das Ergebnis endlos langer Trainingsläufe. Wer das nicht schon vorher gewusst hat, dem führen es die neuronalen Netze deutlich vor Augen. Rechenläufe auf Workstations, die viele Wochen dauern, sind hier durchaus üblich.

Daher kann man die Anwendung neuronaler Netze nicht unabhängig von der Berechnung der Gewichte betrachten. Die Wahl einer bestimmten Netztopologie nützt nichts, wenn sich die Gewichte kaum oder gar nicht bestimmen lassen. In der Praxis entscheidet man sich häufig für einen Kompromiss, d.h. man verwendet eher einfache Netzstrukturen, die sich dann leicht trainieren lassen. Allerdings darf dabei nicht ausser Blick geraten, dass eine künstliche Intelligenz und ihre wünschenswerten Eigenschaften der Abstraktion nur möglich sind, wenn die Sigmoid-Funktion eine möglichst steile Kennlinie besitzt.

Der Backpropagation-Algorithmus hat sich bei vorwärtsgerichteten Netzen als Standard zum Bestimmen der Gewichte durchgesetzt. Zur Zeit gibt es dazu keine Alternative. Neben der Grundform dieses Verfahrens existieren eine Reihe von Erweiterungen, die bei bestimmten Netzen zu einer verkürzten Rechenzeit führen.

4.2 Der Backpropagation-Algorithmus

Dieser Algorithmus basiert auf der Methode des überwachten Lernens („superwised training“). Dazu setzt man zu Beginn per Zufallsgenerator die Gewichte, legt die Eingangssignale an das Netz, berechnet die Ausgangssignale und bildet anschließend die Differenz zu den gewünschten Ausgangssignalen. Diese Differenz entspricht dem Fehler des Netzes. Schließlich verändert man die Gewichte so, daß der Fehler gegen Null konvergiert. Dann beginnt ein neuer Rechenlauf, den man als Trainieren bezeichnet.

Das Verändern der Gewichte, also das Lernen, ist allerdings nicht so einfach, denn es sind bei einer Anzahl von m Eingangssignalen eine Anzahl von m Ausgangsmustern zu trainieren. Hat z.B. das Netz das erste Muster vollständig gelernt, dann darf es beim Trainieren des zweiten Musters auf keinen Fall das erste Muster wieder wegtrainieren (vergessen). Man muß daher die Muster sequentiell etwas antrainieren und dann die Trainingsläufe zyklisch wiederholen.

Das Verändern der Gewichte basiert wieder auf dem biologischen Modell des Lernvorganges im Gehirn. Die Lernregel nach Hebb besagt:

> Wenn zwei aufeinander folgende Neuronen feuern, wird der Gewichtsfaktor, der die beiden Neuronen verbindet, erhöht.

$$\Delta w_{i,j} = \gamma \cdot output_i \cdot output_j \tag{4.2}$$

Die Änderung ist mit einem Faktor γ proportional dem Produkt aus dem Ausgangssignal und dem Eingangssignal des Neurons. γ stellt die Lernrate dar und sollte zwischen 0 und 1 liegen.

Dieser Grundalgorithmus gemäß der Gleichung (4.2), der sich auch für das nicht-überwachte Lernen eignet, dient nun als Ausgangspunkt für das Backpropagation-Verfahren. 4.10 Dazu führt man einen Fehlerterm δ ein, der dem betreffenden Neuron zugeordnet ist. Ausgehend vom Fehler der Ausgangsschicht schreitet man rückwärts durch das neuronale Netz und schreibt (propagiert) den Fehler bis auf die Eingangsschicht zurück. Daher stammt die Bezeichnung „Backpropagation“. Schließlich multipliziert man die Änderung der Gewichte noch mit der Ableitung der Sigmoid-Funktion, um die Konvergenz des Verfahrens zu beschleunigen.

Die Konvergenz dieses Verfahrens ist in jedem Anwendungsfall zu überprüfen. Falls das neuronale Netz eine zu große oder zu kleine Zahl von Neuronen aufweist oder falls die zu trainierenden Muster sich zu sehr ähneln, wirkt sich das ungünstig auf die Konvergenz des Verfahrens aus.

Die Konvergenz des Verfahrens wird für ungünstig dimensionierte Netze durch Nebenminima stark gefährdet. Falls dieser kritische Fall auftritt, führt weder eine Erhöhung noch eine Erniedrung der Gewichte des neuronalen Netzes zu einer Verbesserung der Ausgangssignale. Um ein Entrinnen aus dieser Falle der Nebenminima zu ermöglichen, erhöht man kurzfristig die Lernrate γ. Der bessere Lösungsweg besteht allerdings darin, durch eine entsprechende Anzahl von Testläufen die optimale Neuronenzahl zu bestimmen und insgesamt eine optimale, d.h. auf das zu lösende Problem angepasste, Netzkonfiguration zu finden.

Der Algorithmus sei am Beispiel des zufällig ausgewählten Gewichtsfaktors $w_{5,7}$ erläutert. Dazu wertet man die Gleichungen (4.3) bis (4.7) in der angegebenen Reihenfolge aus und beginnt dann wieder mit der ersten Gleichung (4.3).

$$NettoInput_2 = output_4 \cdot weight_{2,4} + output_6 \cdot weight_{2,6} \tag{4.3}$$

$$\delta_2 = (output_2 - referenz_2) \cdot sigmoid\,'(NettoInput_2) \tag{4.4}$$

$$\delta_5 = sigmoid\,'(NettoInput_5) \cdot (\delta_1 \cdot weight_{1,5} + \delta_2 \cdot weight_{2,5} + \delta_3 \cdot weight_{3,5}) \tag{4.5}$$

$$\Delta weight_{5,7} = \gamma \cdot \delta_5 \cdot output_7 \tag{4.6}$$

$$weight_{5,7}^{neu} = weight_{5,7}^{alt} \tag{4.7}$$

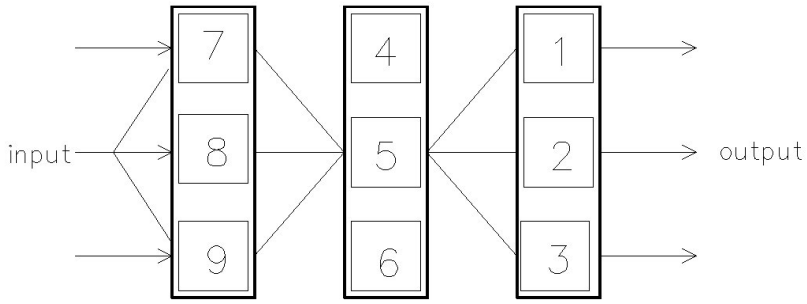


Abb.: 4.10: Beispielhafter Verlauf des Backpropagation-Verfahrens

Als Abbruchkriterium dient eine Fehlerschranke, welche die minimale Übereinstimmung zwischen den output Signalen des Netzes und den zu trainierenden Mustern definiert. Zur Einsparung von Rechenzeit kann man den analytischen Zusammenhang zwischen den Sigmoid-Funktionen und ihren Ableitungen nutzen

$$\frac{1}{1+e^{-x}} \quad \longrightarrow \quad sigmoid\,'(netto) = sigmoid(netto) \cdot (1 - sigmoid(netto)) \tag{4.8}$$

$$\tanh(x) \quad \longrightarrow \quad \tanh\,'(netto) = 1 - \tanh^2(netto) \tag{4.9}$$

und die Sigmoid-Funktion und ihre Ableitung in Potenzreihen mit wenigen Koeffizienten entwickeln.

Nach Abschluß der Trainingsläufe ist das Perceptron-Netz in der Lage, die gespeicherten Muster in der Recall-Phase an den Ausgang zu legen. Die eigentliche Leistungsfähigkeit demonstriert das Netz jedoch in seiner Fähigkeit, neue und von den gespeicherten abweichende Eingangsmuster, wie dies kennzeichnend für die Bilderkennung ist, optimal zu klassifizieren.

Normiert man alle Eingangssignale auf den Zahlenbereich von $\pm$ 1, dann läßt sich bei den einzelnen Trainingsläufen ein erstaunlicher Zusammenhang feststellen.

Die Häufigkeitsverteilung aller Gewichte konvergiert gegen die Gauß´sche Glockenkurve der Normalverteilung.

4.3 Vollständige Neuronenverbindungen

Bei den Perceptron-Modellen handelt es sich um vorwärtsgekoppelte Netze, deren Anwendungsgebiet vor allem bei der Erkennung von Mustern liegt. Die Berechnung der Gewichte erfolgt nach der Backpropagation-Methode und ist für eine Anzahl von bis zu ca. 100 Gewichten im Hinblick auf Konvergenz und Rechenzeit im Prinzip gut handhabbar. Anspruchsvolle Anwendungen lassen sich damit allerdings nicht realisieren.

Orientiert man sich dagegen an den Strukturen biologischer neuronaler Netze, wie z.B. der des Fadenwurms, dann sind Rückkopplungen und Abkürzungen („short cuts“) erforderlich. Als „short cuts“ bezeichnet man Verbindungen, die einen oder mehrere umliegende Zwischenneuronen überspringen. Damit lassen sich dann reflexartige Reaktionen nachbilden.

Die allgemeine Ausgangstopologie verbindet alle Neuronen vollständig untereinander. Stellt man dann beim Trainieren des Netzes fest, dass einige Gewichte sehr klein sind oder sich nicht trainieren lassen, dann entfernt man diese Verbindungen und startet eine neue Trainingsphase. Bei anderen Methoden berechnet man während der Recall-Phasen die Performance des Netzes und schaltet dann in der nachfolgenden Trainingsphase einzelne Neuronen bzw. Verbindungen zu oder ab. Somit entstehen adaptive Topologien, die Voraussetzung einer Maschinen-Intelligenz sind.

Die Begeisterung für diese Verfahren wird leider jedoch stark gedämpft durch die Schwierigkeiten, die sich bei der Berechnung der Gewichte einstellen. Fehlende Konvergenz, Instabilitäten und endlos lange Rechenzeiten haben bisher den Durchbruch dieser Verfahren verhindert. Als Zwischenschritt bietet es sich nun an, die Zahl der Neuronen zunächst klein zu halten, um das Verhalten dieser künstlichen Intelligenzen überhaupt untersuchen zu können. Außerdem sind zur Bestimmung der Gewichte andere Berechnungsverfahren erforderlich, wie z.B. genetische Algorithmen, weil der Backpropagation-Algorithmus für rückgekoppelte Netze nicht ausgelegt ist. Alle diese Probleme lassen sich mit einer Netz-Topologie vermeiden, die im Jahr 1982 von Hopfield vorgeschlagen wurde:

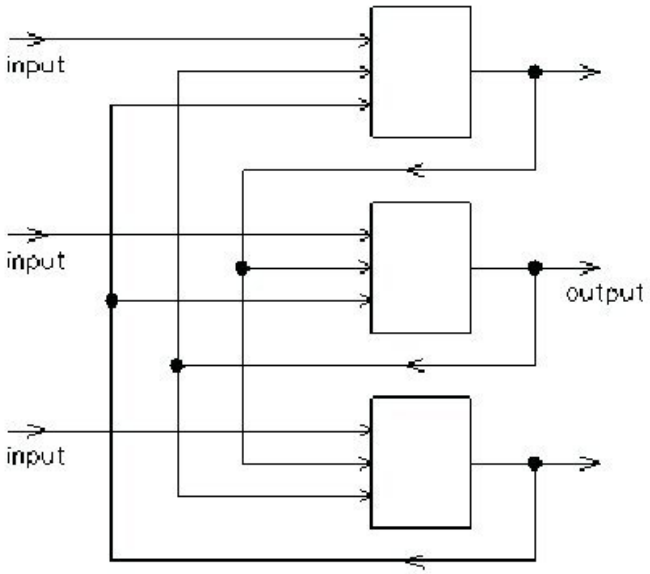


Abb.: 4.11: Struktur eines Hopfield-Netzes

Das Hopfield-Netz besteht aus n Neuronen, nxn symmetrischen Gewichten, n Eingängen und n Ausgängen, das bis auf die Selbstrückkopplung vollständig verbunden ist. Der Gewichtsfaktor für den Input wird fest zu Eins gesetzt.

$$\text{Anzahl der Neuronen} = N \tag{4.10}$$

$$\text{Anzahl der zu speichernden Muster} = M \tag{4.11}$$

$$\text{Gewichte im input Kanal} = 1 \tag{4.12}$$

$$weights_{i,j} = weights_{j,i} \tag{4.13}$$

$$weights_{i,j} = \frac{1}{N}\sum_{i=1}^{i=M}(reference_{output} \cdot Netz_{output} \tag{4.14}$$

$$\text{Anzahl der Gewichte insgesamt} = N(N-1) \tag{4.15}$$

Die Gleichung (4.12) ergibt sich zwangsläufig, da in der Gleichung (4.14) die Reihenfolge der Multiplikation vertauschbar ist.

Durch diese einschränkenden Bedingungen erhält man ein Netz, das mittels der Gleichung (4.14) eine geschlossene Berechnung der Gewichte ermöglicht und damit jedes Konvergenzproblem vermeidet. Die Auswertung der Gleichung (4.14) erfolgt einmalig beim Programmstart, was Echtzeit-Anwendungen unterstützt.

Allerdings dauert es nach dem Anlegen der Eingangssignale mehrere Schritte, bis das Hopfield-Netz einen stabilen Zustand erreicht, denn die rückgekoppelten Ausgangssignale der einzelnen Neuronen beeinflussen im nächsten Schritt wieder das eigene Ausgangssignal. In diesem Zusammenhang ist es günstig, bei der Berechnung der Ausgangssignale abwechselnd nach dem Zufallsprinzip die einzelnen Neuronen auszuwählen und dafür das Ausgangssignal zu berechnen.

Im Anwendungsfall der Gesichtserkennung demonstriert das Hopfield-Netz seine enorme Leistungsfähigkeit. So kann eine Person bei einem Bildscan z.B. eine Brille tragen, den Kopf geneigt halten und sich in einem zufälligen Abstand zur Kamera befinden. Das Hopfield-Netz liefert dann im besten Fall ein Ausgangsbild, das mit dem gespeicherten Bild exakt übereinstimmt, d.h. es filtert die Verfälschungen des Bildscans heraus. Darin liegt die eigentliche Stärke des Hopfield-Netzes.

Beispiel 4.1:

Das folgende Beispiel zeigt die Ergebnisse für ein Hopfield-Netz mit 400 Neuronen, deren Eingangssignale einer 20x20 Pixelmatrix entsprechen, von Hopfield selbst publiziert wurde und unter der Bezeichnung „Mützenmann“ großes wissenschaftliches Aufsehen erregte.

In der Abb. 4.12 liegt nur das Bild der Mütze am Netz an. Bereits nach zwei Schritten generiert das Netz das gespeicherte Orginalbild.

In der Abb. 4.13 enthält das Eingangsbild 30% Rauschanteil und das graphische Objekt ist für das menschliche Auge nicht mehr sichtbar. Auch in diesem Fall liefert das Netz in zwei Schritten das fehlerfreie Ergebnis.

Erst ab einem Rauschanteil von 40% ist das Netz überfordert und liefert, wie in der Abb. 4.14 dargestellt, kein sinnvolles Ergebnis mehr.

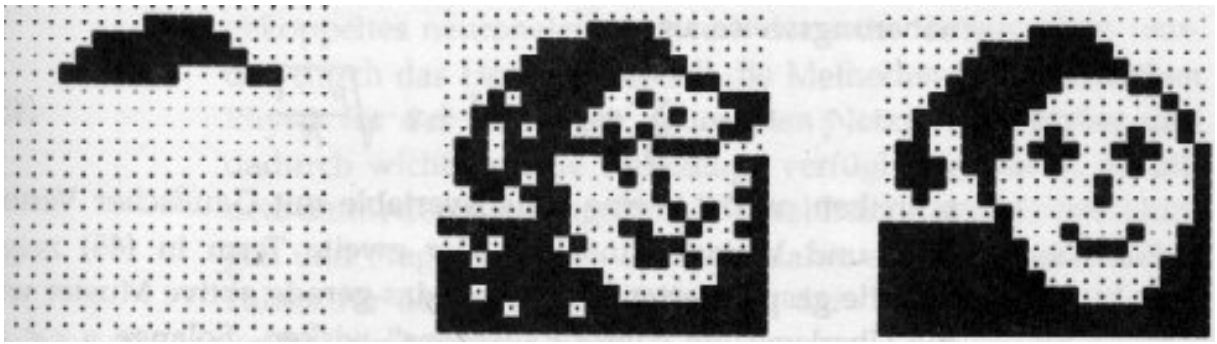


Abb.: 4.12: Oberer Mützenteil als Eingabe, 1. und 2. Rechenschritt

Abb.: 4.13: Komplettes, 30% verrauschtes Bild als Eingabe, 1. und 2. Rechenschritt

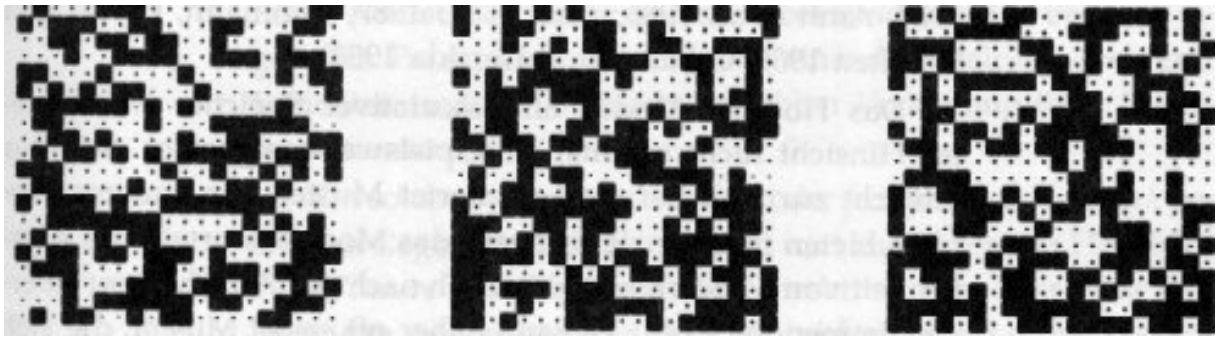


Abb.: 4.14: Komplettes, 40% verrauschtes Bild als Eingabe, 1. und 2. Rechenschritt

Beispiel 4.2:

Gegeben sei ein elementares Hopfield-Netz, das nur aus den beiden Neuronen i und k besteht:

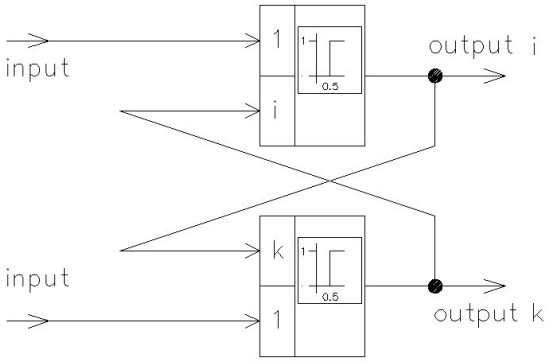


Abb.: 4.15: Beispiel eines Hopfield-Netzes mit zwei Neuronen

Dieses Netz soll zwei Muster speichern:

Referenz-Muster 1: $i = 0 \quad k = 0$
Referenzl-Muster 2: $i = 1 \quad k = 1$

In diesem einfachen und anschaulichen Fall lassen sich die beiden Koeffizienten i und k gemäß der Summenformel (4.13) leicht berechnen:

$$i = \frac{0 \cdot 0 \ + \ 1 \cdot 1}{2} = 0.5$$
$$k = \frac{0 \cdot 0 \ + \ 1 \cdot 1}{2} = 0.5$$

Man sieht, dass immer gelten muss i = k, denn die Reihenfolge der Multiplikationen werden bei i und k nur vertauscht. Diese Eigenschaft erweist sich im Hinblick auf den erforderlichen Rechenaufwand als sehr nützlich. Die Gewichte für die Eingangssignale setzt man auf 1. Als Schaltfunktion dient ein Hard-Limiter, der bei netto > 0.5 den output von 0 auf 1 umschaltet:

$$output = \begin{cases} 0 & \text{für} \quad \text{netto} \leq 0.5 \\ 1 & \text{für} \quad \text{netto} > 0.5 \end{cases}$$

Legt man die Ausgangsmuster an den Eingang, dann stellen sich am Ausgang die gewünschten Signale ein. Dieser Test ist erforderlich, um die Berechnung der Gewichte zu testen.

Interessant wird es, wenn am Hopfield-Netz nun neue Muster anliegen.

Scan-Muster 1: $i = 0 \quad k = 1$
Scan-Muster 2: $i = 1 \quad k = 0$

Wie ein Blick auf die Abb. 4.15 zeigt, überwiegt jeweils das Signal, das gerade den Wert 1 annimmt. Das Netz schaltet daher die angelegten Scan-Muster auf den Ausgang durch.

Damit demonstriert das einfachste Hopfield-Netz mit seinen zwei Neuronen bereits

die Fähigkeit zur Abstraktion

Am Ausgang des Hopfield-Netzes treten alle vier Muster auf, obwohl die Berechnung der Gewichte nur auf zwei Mustern basierte.

4.4 Testläufe eines Hopfield-Netzes

Die folgenden Beispiele demonstrieren die grundlegenden Eigenschaften eines Hopfield-Netzes an überschaubaren Anwendungen.

Beispiel 4.3:

Das Beispiel 4.3 demonstriert für den einfachen Fall einer Spielkarten-Erkennung die grundsätzliche Wirkungsweise eines Hopfield-Netzes.

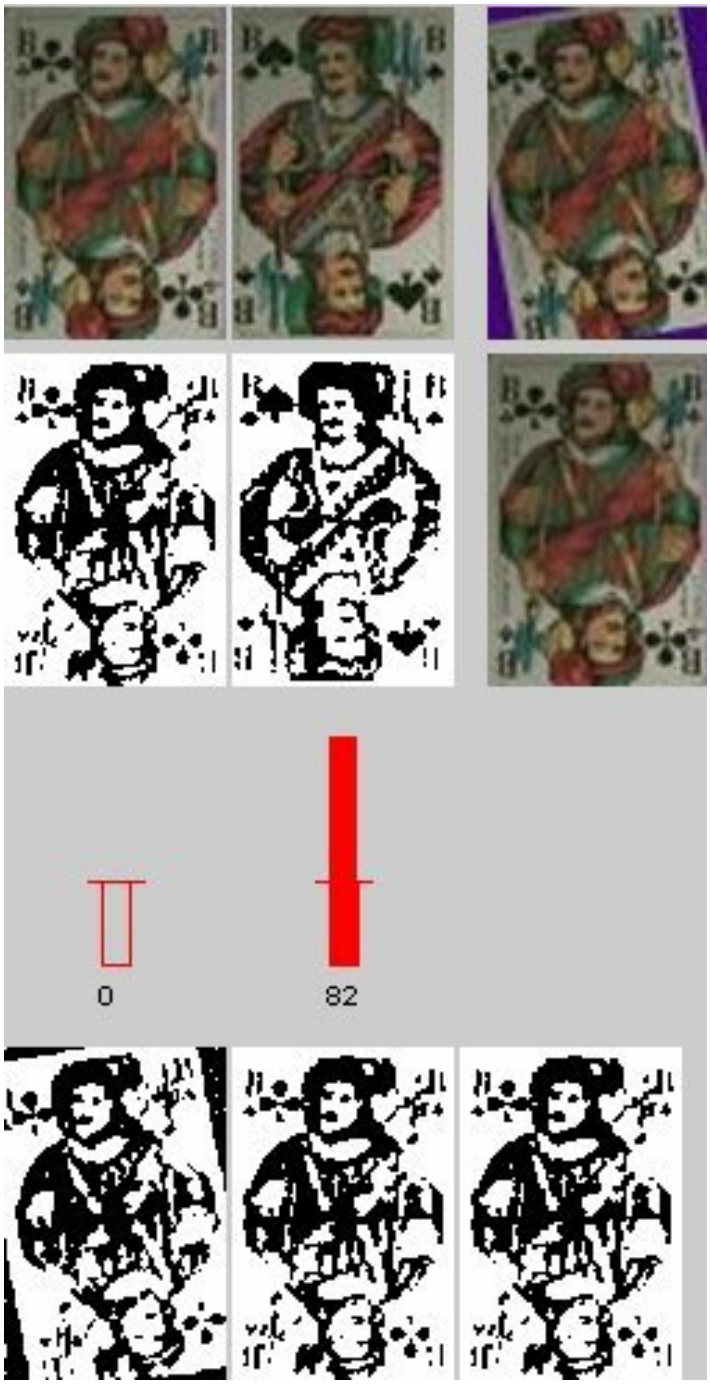


Abb.: 4.16: Erkennung einer gedrehten Spielkarte mit dem Hopfield-Netz

Die Abb. 4.16 zeigt in der linken und mittleren Darstellung der obersten Zeile zwei Spielkarten als Referenz-Pixelbilder, nämlich einen „Kreuz-“ bzw. einen „Pik-Buben“. Der oben rechts dargestellte gedrehte „Kreuz-Bube“ wurde eingescannt und soll nun vom Hopfield-Netz erkannt werden.

Zur Berechnung der Gewichte des Hopfield-Netzes ist eine binäre Beschreibung der beiden Referenz-Pixelbilder erforderlich. Alle Pixel, die unterhalb des Mittelwertes aller Pixel liegen, nehmen den Wert -1 an bzw. werden umgekehrt auf den Wert 1 gesetzt. Schwarze Pixel entsprechen dem Pixelwert -1 und weiß ist dem Pixelwert 1 zugeordnet. Durch die Algorithmen der Vorfilterung sind die Abbildungen grundsätzlich mittelwertfrei und daher genügt bei der Bestimmung der diskreten Werte eine schnelle Vorzeichenbetrachtung.

Die Graphiken, mit denen die Gewichte des Hopfield-Netzes berechnet werden, lassen sich in der Abb. 4.16 unterhalb der beiden farbigen Referenzbilder erkennen.

Legt man an das Hopfield-Netz nun das scharz-weiße Bild (siehe Graphik links unten) der eingescannten Spielkarte an, dann erscheint bereits nach dem ersten Schritt am Ausgang des Hopfield-Netzes die eingespeicherte und nun zurück gedrehte Graphik des entsprechenden Referenzbildes, das sich dann im zweiten (siehe unterste mittlere Graphik) und dritten Rechenschritt (siehe unterste rechte Graphik) als stabil erweist. Dies ist eine beeindruckende Leistung des Hopfield-Netzes, denn die Drehrichtung des gescannten Bildes war unbekannt und wurde innerhalb der Algorithmen des Hopfield-Netzes auch nirgends berechnet.

Die Fehlerwerte sind in Abb. 4.16 durch rote Säulen visualisiert. Es ist zu erkennen, das der eingescannte, gedrehte „Kreuzbube“ fehlerfrei erkannt wird, während dagegen die Übereinstimmung zum „Pik-Buben“ einen hohen Fehlerwert von 82% aufweist.

Die kurzen waagerechten Linien in der Säulendarstellung markieren einen angenommenen Schwellwert von 30 % für den dargestellten Fehlerwert. Falls der minimale Fehlerwert unterhalb dieses Schwellwertes liegt, gilt der Erkennungsalgorithmus als erfolgreich und das entsprechende Originalbild wird in der Graphik in der zweiten Zeile ganz rechts dargestellt.

Für die Erkennung unter Echtzeitbedingungen genügt es, dem gescannten Bild in der obersten Reihe rechts das erkannte Bild aus der Bildersammlung, das rechts in der zweiten Reihe zu sehen ist, gegenüber zu stellen. Falls der Fehlerwert den Schwellwert jedoch überschreitet, erscheint auf dem Kontrollbildschirm statt einem Bild aus der Bildersammlung eine entsprechende Fehlermeldung, an der dann ersichtlich ist, dass keine Person mit ausreichender Sicherheit erkannt werden konnte.

Der Zahlenwert des unterhalb der Säulen geplotteten Fehlers basiert auf der Differenzbildung der logischen Pixelwerte zwischen dem Ausgangsbild des Hopfield-Netzes und dem erkannten Originalbild, dessen Graustufenwerte auf die logischen Werte umzurechnen sind. Der absolute maximale Fehlerwert, der als Basis zur Normierung der Fehlerwerte dient, beträgt daher bei N Pixeln exakt 2 N.

Die dargestellten beiden Spielkarten besitzen eine Auflösung von jeweils 80x120 Pixeln. Das entsprechende Hopfield-Netz erfordert daher

$$(80 \cdot 120)^2 = 9600^2 \approx 92 \text{ Millionen}$$

Gewichte, die sich jedoch geschlossen innerhalb einer Sekunde berechnen lassen. Allerdings kann der dafür nötige Speicherplatz Sorgen bereiten, denn selbst bei einer Darstellung der Gewichte im Format „byte“ sind ca. $92/2 = 46$ MByte im Hauptspeicher zu halten. Die Halbierung des Speicherbedarfs verdankt man der Symmetrie der Gewichte in einem Hopfield-Netz. Bei größeren Bildern steigt der Speicherbedarf extrem an und ist dann kaum noch zu beherrschen. Die Vergrößerung des Bildes auf 160x240 Pixel führt bereits zu einem 16-fachen Speicherbedarf von ca. 1,4 GByte und lässt sich daher unter Echtzeitbedingungen nicht mehr realisieren.

Beispiel 4.4:

Die Abb. 4.17 stellt die Erkennung einer Person dar, welche vor der Überwachungskamera die Hand zum Gruß an die Stirn legt (rechtes Bild in der ersten Zeile). Als Referenzbilder dienen die vier Gesichtsaufnahmen der ersten Zeile. Das Hopfield-Netz versucht diese Hand auszuradieren, was bis auf den in der linken Säule dargestellten und unterhalb des Grenzwertes liegenden Fehlerwert von 33 weitgegend auch gelingt. Die Fehlerwerte der anderen Bilder liegen deutlich höher und damit konnte dieses Gesicht sicher erkannt und in der Graphik ganz rechts in der zweiten Zeile als Orginalbild dargestellt werden. Das Hopfield-Netz konvergiert nach wenigen Schritten, wie an der Bildfolge in der untersten Zeile zu sehen ist.

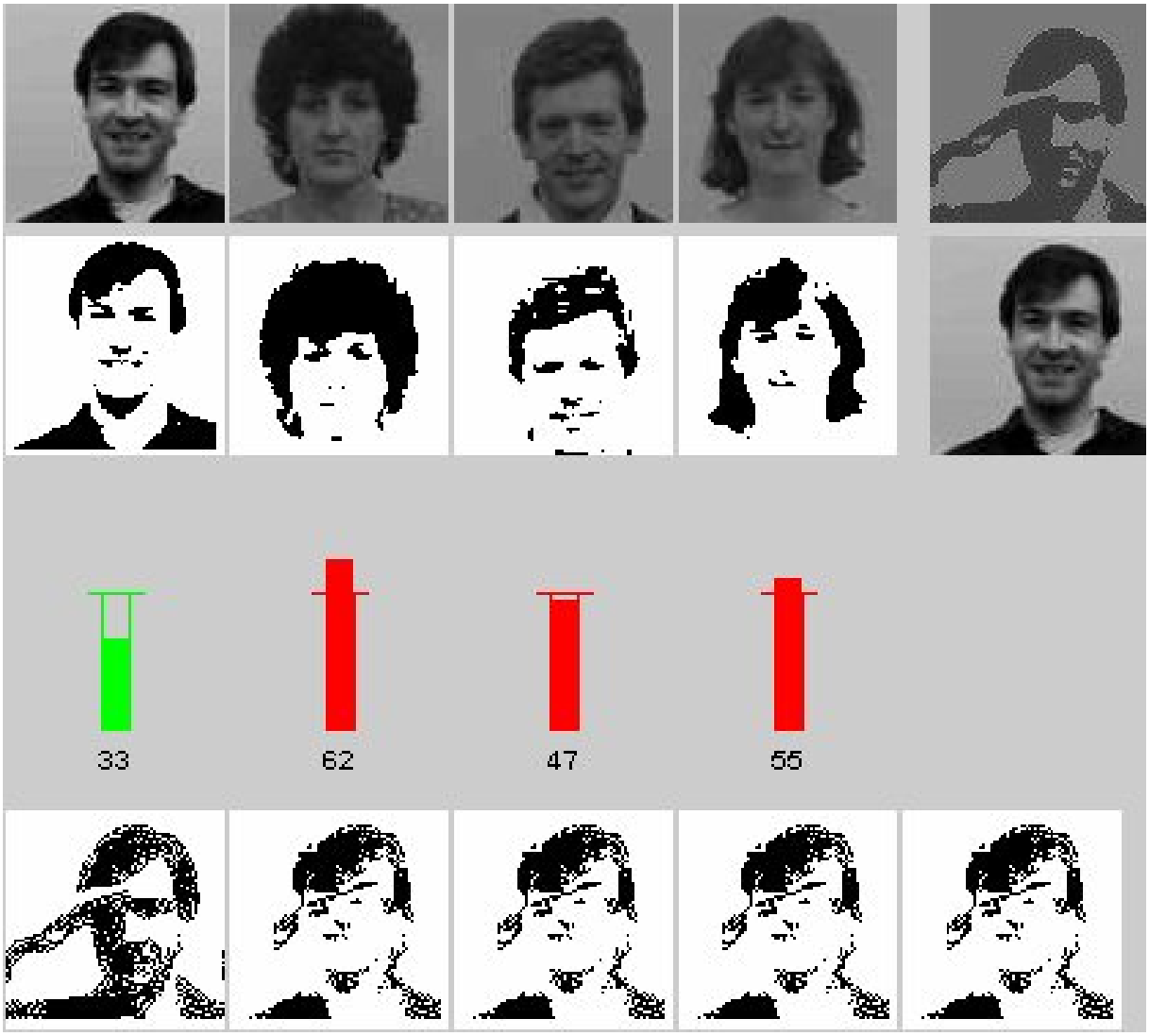


Abb.: 4.17: Beispiel einer Gesichtserkennung (Mann mit einer Hand an der Stirn)

Die in der Abb. 4.18 gezeigten Ergebnisse sind besonders beeindruckend, denn das Hopfield-Netz war in der Lage, die Person trotz ihrer dunklen Brille mit 19 Fehlereinheiten sicher zu erkennen. Bereits nach dem ersten Schritt ist die Brille im Ausgangssignal des Hopfield-Netzes verschwunden und das Bild weist mit dem Referenzbild eine hohes Maß an Überstimmung auf.

Im Beispiel der Spielkarten dienten 50x80 Pixelbilder und im Fall der Gesichtsaufnahmen 80x80 Pixelbilder als Ausgangsbasis. Bei einem 80x80 Pixelbild besteht das Hopfield-Netz aus einer Anzahl von N = 6400 Neuronen, von denen jedes einzelne Neuron die Ausgänge der anderen 3599 Neuronen und das eigentliche Eingangssignal aufnimmt. Somit folgen insgesamt

$$\begin{aligned} \text{Anzahl der Gewichte} &= N \cdot N = (80\text{x}80) \cdot (80\text{x}80) = 3600^2 = 12.960.000 \\ &\longrightarrow \quad 51,84 \text{ MByte beim Java-Datentyp float} \end{aligned}$$

Daher sollte man bei der Programmierung den Datentyp double vermeiden und besser mit dem Type float rechnen. Bei der Entwicklung des Erkennungssystems, das in diesem Bericht ab dem Kapitel 7 beschrieben wird, gelingt es durch eine numerische Umformung der Hopfield-Gleichungen auf den Datentyp byte auszuweichen und dadurch die Datenmenge auf ca. 12 MByte zu begrenzen.

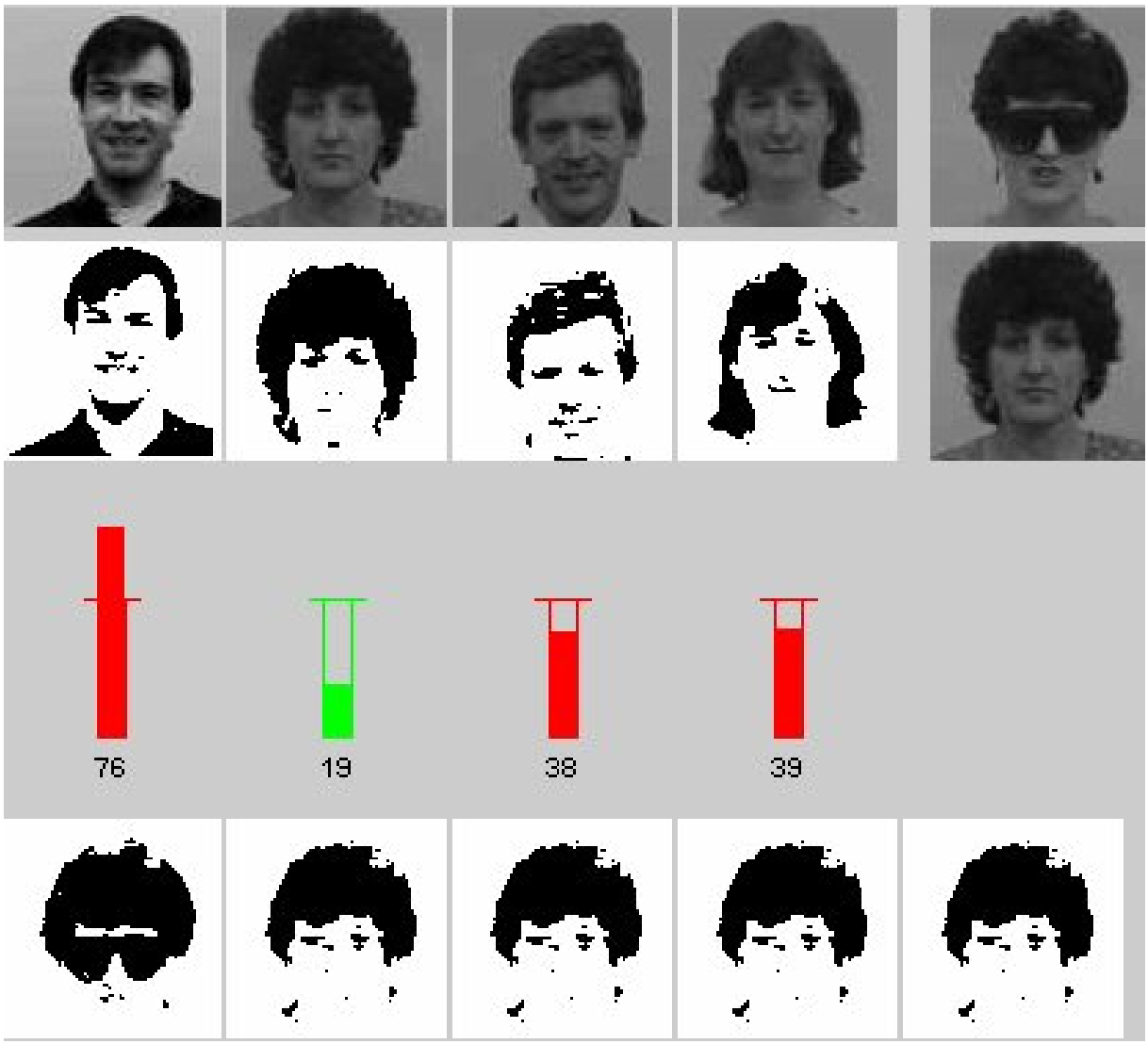


Abb.: 4.18: Beispiel einer Gesichtserkennung (Frau mit Brille)

Das Beispiel „Frau mit Brille“ demonstriert besonders anschaulich das große Potential des Hopfield-Netzes zur Bilderkennung, denn dieser echtzeitfähige Algorithmus verfügt über erstaunliche Eigenschaften, die sich bei der Erkennung vermummter Personen als sehr nützlich erweist:

- Das neuronale Hopfield-Netz eliminiert graphische Objekte, die nicht im Referenzbild, sondern nur in der gescannten Aufnahme vorhanden sind.
- Das neuronale Hopfield-Netz kompensiert die geometrischen Transformationen einer gescannten Aufnahme und stellt dadurch das Referenzbild wieder her.

Kapitel 5

Matching von Bildpunkten

5.1 Grundlagen

Der Begriff „Matching“ bezieht sich auf Methoden, welche eine Auswahl von Bildpunkten eines gespeicherten Referenzbildes mit den entsprechenden korrespondierenden Bildpunkten eines gescannten Bildes abgleichen, d.h. „matchen“. Der Abgleich erfolgt in der Regel auf der Basis von geometrischen Transformationen der Bildpunkte des gescannten Bildes. Nimmt die Summe der betrags- oder quadratisch aufsummierten Differenzen zwischen den Bildpunkten des Referenzbildes und des geometrisch transformierten gescannten Bildes ein Minimum an, dann läßt sich dies, über die Übereinstimmung der Bildpunkte hinaus, auch als eine inhaltliche Übereinstimmung beider Graphiken interpretieren. Diese Schlussfolgerung setzt allerdings voraus, dass die Bildpunkte einen hohen Informationsgehalt besitzen und sich das zu erkennende graphische Objekt durch Punktfolgen, wie z.B. Randlinien, näherungsweise beschreiben läßt.

Der Ablauf eines Matching-Verfahren gliedert sich in drei Stufen

1. Gewinnung von Bildpunkten, z.B. durch die Berechnung von Kantenlinien
2. Abgleich der Bildpunkte des gescannten Bildes auf die Referenz-Bildpunkte
3. Ausgabe des Originalbildes, falls die minimale Fehlerdifferenz zwischen den Bildpunkten des Referenzbildes und den Bildpunkten des transformierten gescannten Bildes unterhalb eines Schwellwertes liegt.

Für das Matching-Verfahren gelten die folgenden Grundbegriffe:

$$\text{Anzahl der Bildpunkte des Referenzbildes} = r \tag{5.1}$$

$$\text{Matrix der Bildpunkte des Referenzbildes} = RefMatrix \tag{5.2}$$

$$\text{Anzahl der Bildpunkte für das gescannte Bild} = s \tag{5.3}$$

$$\text{Matrix der Bildpunkte des gescannten Bildes} = ScanMatrix \tag{5.4}$$

$$\text{Anzahl der Bildpunkte für das Matchingverfahren} = p \tag{5.5}$$

$$\text{Matrix der Bildpunkte des gescannten und transf. Bildes} = TransMatrix \tag{5.6}$$

$$\text{Korrespondenz-Matrix} = CMatrix \tag{5.7}$$

$$p = \text{minimum}(r, s) \tag{5.8}$$

$$\text{Differenz-Matrix } diffMatrix = RefMatrix - TransMatrix \tag{5.9}$$

$$\text{Fehlerwert} f = \frac{1}{2p} \sum_{i=1}^{p} \sum_{j=1}^{2} |diffMatrix(i,j)| \tag{5.10}$$

Die Matrizen bestehen für 2-dimensionale Graphiken aus zwei Spalten und die Anzahl der Zeilen entsprechen der jeweiligen Anzahl von Punkten. Der Wert von p des Matchingverfahrens stimmt mit der kleineren Anzahl der Werte r, s überein, denn für die Berechnung

der Differenz-Matrix müssen die Dimensionen übereinstimmen. Die $CMatrix$ stellt den gesuchten Zusammenhang zwischen der Reihenfolge der Punkte dar und fungiert wie eine Drehscheibe wenn es darum geht, die $ScanMatrix$ in die $TransMatrix$ zu verwandeln.

Für die Korrespondenz-Matrix ergeben sich insgesamt p! Varianten, d.h.für $p = 6$ folgen $6! = 720$ Möglichkeiten. Für eine geringe Anzahl p von Punkten besteht die Möglichkeit, alle möglichen Varianten der $CMatrix$ anzuwenden. Nimmt f einen absoluten minimalen Wert an, dann gilt die entsprechende $CMatrix$ als richtig. Falls der Fehlerwert f für alle Varianten der $CMatrix$ nicht unterhalb eines Schwellwertes absinkt, dann besteht keine ausreichende Übereinstimmung zwischen den Matrizen $RefMatrix$ und $TransMatrix$.

Bei einer großen Anzahl p ist ein rechenintensives „non-rigid point matching“ Problem zu lösen für das allgemeine Lösungsverfahren leider nicht existieren. Eine Echtzeitfähigkeit der Algorithmen, die für eine Gesichtserkennung erforderlich ist, läßt sich auch mit speziellen Näherungsverfahren des „non-rigid point matching“ Problems kaum noch realisieren.

Die nachfolgend beschriebenen Matching-Verfahren basieren auf einer Anzahl von $p \leq 6$ Punkten. Die Rechenzeiten liegen unter diesen Bedingungen noch unterhalb einer Sekunde. Eine Java-Methode liefert alle Kombinationen der Punktfolge zurück. Die Abb. 5.1 zeigt für einen Linienzug mit 5 Punkten alle möglichen 120 Verläufe:

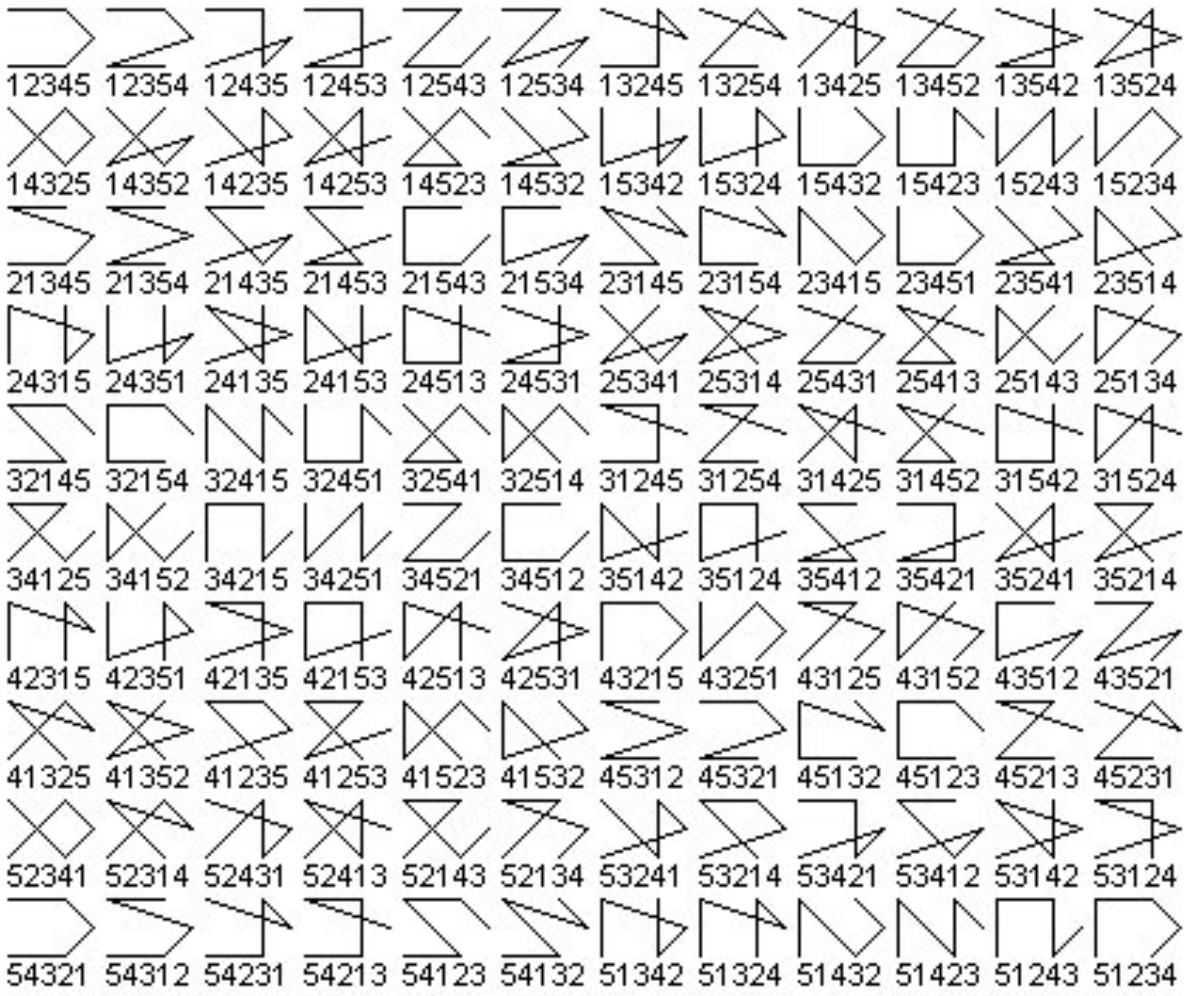


Abb.: 5.1: 120 Permutationen einer Kantenlinie mit 5 Punkten

5.2 Basis-Algorithmus der Hough-Transformation

Die Hough-Transformation gehört zu der Gruppe der Parameter-Transformationen. Eine Parameter-Transformation ist in der Lage, aus einer graphischen Darstellung im Pixelformat einen Parametersatz zu liefern, der das graphische Objekt beschreibt. Mit diesen Parametern und den dazu gehörenden Gleichungssystemen kann man dann das graphische Objekt neu erzeugen. Als Parameter eignen sich vor allem Abstände und Winkel.

Das bekannteste Beispiel stellt in diesem Zusammenhang das graphische Objekt einer Geraden dar.

$$\text{Geradengleichung:} \quad y(x) = a \cdot x + b \tag{5.11}$$

Berechnet man aus einer Punktfolge, die auf der Geraden liegen, die Steigung a und den Schnittpunkt b mit der senkrechten Achse, dann liegen zwei Parameter vor, mit denen die Gerade bis auf ihren Definitionsbereich erzeugt werden kann. Aus der Graphik beliebig ausgewählte Pixelpaare i liefern dann ein Parameterpaar a_i, b_i zurück. Alle Pixelpaare mit identischen Parametersätzen liegen somit auf einer Geraden. Ein Zähler („Akkumulator“) hält fest, wie oft ein bestimmter quantisierter Parametersatz aufgetreten ist. Erreicht der Akkumulator für einen speziellen Parametersatz z.B. den Wert 8, dann liegen genau 8 Punkte auf der Geraden, die durch diesen Parametersatz definiert sind.

Die Definitionsbereiche der Parametersätze sind häufig bekannt, denn sie beschreiben die maximale Ausdehnung der graphischen Objekte. Dadurch ist es möglich, den Suchbereich zu quantisieren und stark einzuschränken. Mit der umgestellten Geradengleichung

$$a_i = y - b_j \cdot x \quad \text{mit} \quad a_{min} \leq a_i \leq a_{max} \quad \text{mit} \quad b_{min} \leq b_j \leq b_{max} \tag{5.12}$$

bestimmt man nun für alle möglichen Parameterwerte b_j diejenigen a_i-Werte, welche an der Stelle (x,y) die Geradengleichung erfüllen. Falls einer der berechneten Werte außerhalb des Definitionsbereiches von a, b liegt, wird der entsprechende Parametersatz verworfen.

Programmtechnisch betrachtet läuft j als Integer zwischen 0 und einer maximalen Anzahl $jmax$. Die entsprechenden kontinuierlichen Werte der Parameter folgen damit zu:

$$b_j = b_{min} + \frac{j}{jmax}\,(b_{max} - b_{min}) \;\; ; \quad 0 \leq j \leq jmax \tag{5.13}$$

$$a_i = y - b_j \cdot x \tag{5.14}$$

$$i = \text{ganzzahliger Wert von} \left(\frac{a_i - a_{min}}{a_{max} - a_{min}}\; imax \right) \tag{5.15}$$

Aus dem Wertepaar einer Pixelposition (x,y) ensteht nun ein ganzzahliges Wertepaar (i,j), das für alle Pixelpositionen, die auf einer Geraden liegen, identische Werte annimmt, denn alle Punkte auf dieser Geraden sind durch die Parameter (a,b) definiert. In der Akkumulator-Matrix A mit den Dimensionen $imax, jmax$ erhöht man an der Stelle i, j den Wert um 1.

Falls z.B. alle n Punkte auf einer Geraden liegen, dann weist die A-Matrix an einer bestimmten Stelle den Wert von n auf und ist sonst überall Null. Enthält die Pixelgraphik eine Anzahl von m unterschiedlichen Geraden, dann beobachtet man in der A-Matrix die ensprechenden „Inseln“, an denen von Null verschiedene Elemente auftreten. Bei verrauschten Pixelgraphiken enthalten diese „Inseln“ dann nur einen geringen Zählerstand, weil zwei oder drei Pixelpunkte zufällig auf einer Geraden liegen können. Für die weitere Auswertung der A-Matrix eignen sich nur die „Inseln“ mit hohem Zählerstand.

Beispiel 5.1:

Das folgende Beispiel basiert auf einer angenommenen Geraden, die in der Abb. 5.2 dargestellt ist:

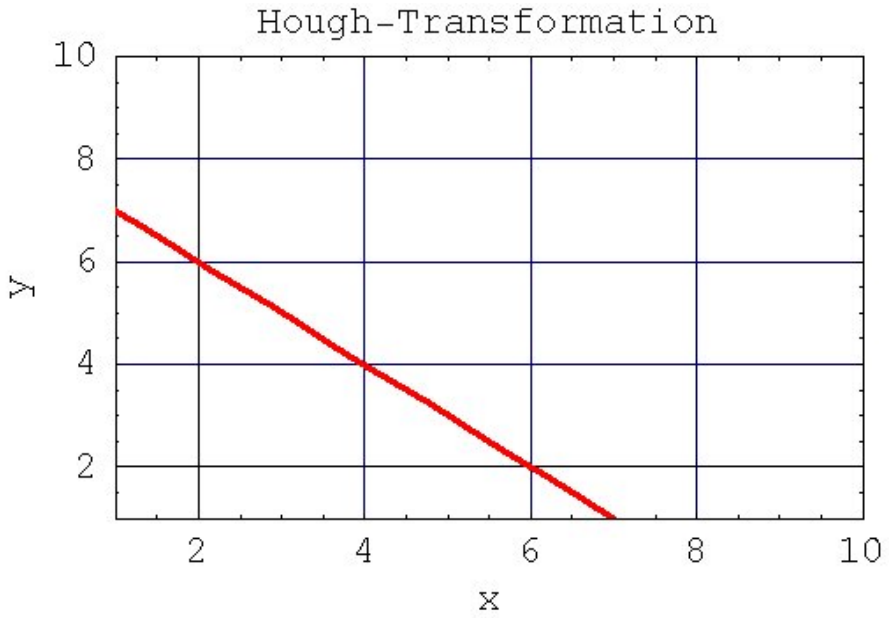


Abb.: 5.2: Geradengleichung des Pixelbildes

Gemäß dem Verlauf der roten Geradenlinie ergeben sich die Definitionsbereiche für die Hough-Transformation zu

$$\begin{aligned} -10 \leq\ & a & \leq 10 \\ -10 \leq\ & b & \leq 10 \\ 0 \leq\ & i & \leq 20 \\ 0 \leq\ & j & \leq 20 \end{aligned}$$

Dies entspricht der folgenden Pixelmatrix, in der die Einsen die roten Punkte kennzeichnen.

$$\text{Pixelbild} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Abb.: 5.3: Pixelbild mit einer einzelnen Geraden

Nach Abschluss der Hough-Transformation betrachtet man die resultierende Akkumulator-Matrix und sucht nach „Inseln“ in dem „Meer“ von Nullen:

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Abb.: 5.4: Akkumulator-Matrix

An der Stelle $i = 8$ und $j = 11$ erkennt man eine 8, die den 8 auf der angenommenen Geraden liegenden Punkten des Pixelbildes 5.3 entspricht.

Mit den zu der Stelle (8,11) gehörenden Parametern a, b folgt wieder die in der Abb. 5.2 dargestellte Gerade.

In der Akkumulator-Matrix 5.4 taucht an einzelnen Stellen die Ziffer 1 auf. Dies ist plausibel, denn jeder einzelne der 8 Punkte kann auf einer zweiten Geraden liegen, wobei diese spezielle scheinbare „Gerade“ nur aus einem Punkt besteht.

Falls das Pixelbild einen überlagerten Rauschanteil enthält, dann wären weitere von Null verschiedene Ziffern in der Akkumulator-Matrix 5.4 aufgetreten.

Bei realen Bildern kommt es darauf an, die relativ wenigen Bereiche maximaler Zahlenwerte innerhalb der Akkumulator-Matrix zu lokalisieren. Gegenüber den vielen Nullen einer Akkumulator-Matrix heben sich die Bereiche mit den größeren Zahlen wie Inseln ab.

Beispiel 5.2:

Zur Demonstration der erstaunlichen Leistungsfähigkeit von Parameter-Transformationen soll nun eine Ellipse innerhalb eines stark gestörten Pixelbildes erkannt werden. Zur mathematischen Beschreibung der Ellipsen eignet sich die Parameterdarstellung einer Ellipse besonders gut:

$$\begin{aligned} i &= r_i \cdot \cos(2\pi t) + m_i \\ j &= r_j \cdot \sin(2\pi t) + m_j \end{aligned}$$

r_i und r_j beschreiben die Radien in i- und j-Richtung und m_i, m_j enthalten die Mittelpunktskoordinaten. Gemäß dieser Beschreibungsform ist die Ellipse durch die folgenden vier Parameter gekennzeichnet:

$$r_i \quad r_j \quad m_i \quad m_j$$

Die Definitionsbereiche dieser Parameter ergeben sich unmittelbar aus der Größe des Pixelbildes hmax und wmax, denn die Ellipse soll das Graphikfenster nicht überschreiten.

$$\begin{aligned} m_i + r_i &\leq hmax \\ m_i - r_i &\geq 0 \\ m_j + r_j &\leq wmax \\ m_j - r_j &\geq 0 \end{aligned}$$

Die Anzahl der Parameter wird im Hinblick auf den Speicher- und Rechenzeitbedarf auf die Fenstermaße hmax und wmax bezogen:

$$\begin{aligned} r_{imax} &= \frac{hmax}{2} \\ r_{jmax} &= \frac{wmax}{2} \\ m_{imax} &= hmax \\ m_{jmax} &= wmax \end{aligned}$$

Die Abb. 5.5 stellt im linken Bild die angenommene Punktfolge einer Ellipse dar. Dieses graphische Objekt wird im mittleren Bild mit Rauschen überlagert. Damit die Ellipse zur Anschauung gerade noch sichtbar bleibt, sind die Ellipsenpunkte etwas dicker gezeichnet. Im rechten Bild sieht man, wie die Hough-Transformation in der Lage ist, die Ellipsenparameter richtig zu berechnen. Die Parameter der angenommenen Ellipse und der aus der Punktwolke berechneten Parameter stimmen exakt überein.

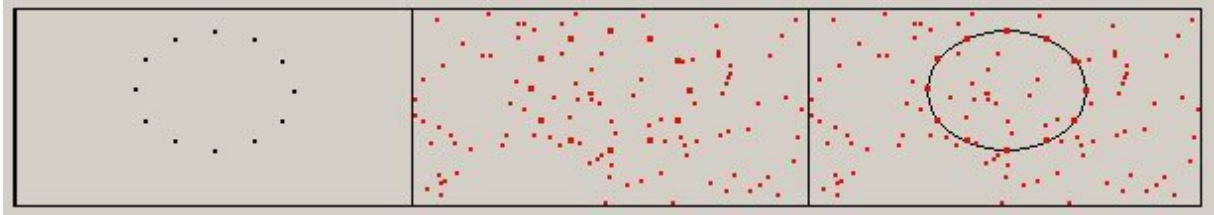


Abb.: 5.5: Erkennen einer Ellipse in einer Punktwolke

Für die dargestellten Fenstermaße von 200x100 Pixeln und die oben angegebenen jeweils 20 Quantisierungsstufen in beiden Achsen ergeben sich Rechenzeiten von drei Sekunden. Damit ist dieser Basisalgorithmus einer Parameter-Transformation für Echtzeit-Anwendungen kaum geeignet. Weitere Parameter, wie etwa eine Schräglage der Ellipse, treiben den Rechenaufwand enorm in die Höhe. Mit der allgemeinen Hough-Transformation, die im nächsten Abschnitt erläutert wird, lässt sich die Leistungsfähigkeit dieser Parameter-Transformation erheblich steigern und damit erst für die Anwendung in der Bilderkennung einsetzen.

5.3 Die allgemeine Hough-Transformation

Wendet man den Basis-Algorithmus der Hough-Transformation auf praxisrelevante Pixelbilder an, dann treten bei der Implementierung ernsthafte Probleme auf:

- Die Definitionsbereiche der Parameter können extrem groß sein, was zu einem enormen Speicheraufwand für die Akkumulator-Matrix A führt.
  (z.B. bei einer Geradengleichung: $y = mx + b \quad \longrightarrow \quad -\infty \leq m \leq \infty$
- Die gesuchte Form liegt nicht als Funktionsgleichung, sondern nur als Punktfolge oder Pixelwolke vor.

Als mögliche Auswege bieten sich die beiden folgenden Berechnungsmethoden an:

- Aufteilung der A-Matrix in zwei Teilmatrizen A1 und A2, wobei gilt:

  $\text{A1}: \quad -1 \leq \text{Parameterwert} \leq 1 \qquad \text{A2}: \quad -1 \leq \frac{1}{\text{Parameterwert}} \leq 1$
- Beschreibung der zu erkennenden Funktionen auf der Basis von Abständen, Winkeln, Gradienten und in der Hesseschen Normalform.

Die Aufteilung der Steigung m von Funktionsverläufen in zwei Teilbereiche begrenzt den Wertebereich auf ± 1, was sich für die Quantisierung der Parameter als sehr günstig erweist.

Die Leistungsfähigkeit der Parameter-Transformation steigt erheblich an, wenn man als Parameter nur noch Abstände und Winkel wählt. Ausgehend von einem frei wählbaren Punkt im Inneren des graphischen Objektes, definiert die Abb. 5.6 die Parameter der allgemeinen Hough-Transformation. Die Berechnung der Winkel basiert auf der Differenz zwischen dem momentan betrachteten Pixelpunkt p_i und den Pixelpunkten, die vor bzw. nach dem Punkt p_i auf der Kantenlinie liegen.

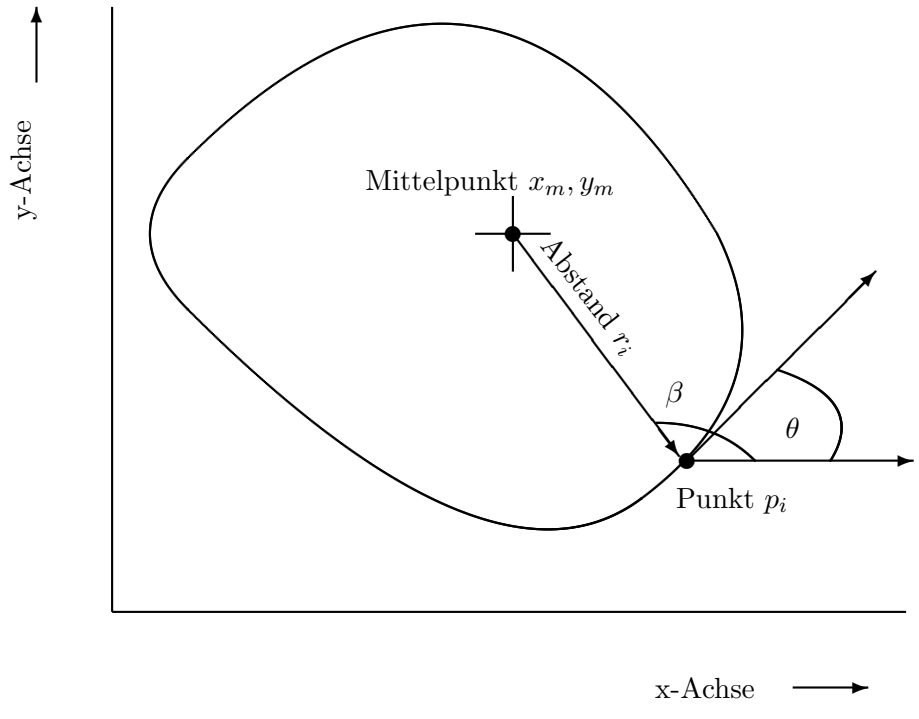


Abb.: 5.6: Die Parameter der allgemeinen Hough-Transformation

Jeder beliebige Punkt p_i in der Graphik ist dann durch die folgenden drei Parameter bestimmt:

- Abstand r von einem Punkt innerhalb des graphischen Objektes
- Winkel β_i der Tangente an einem Punkt p_i
- Betrag des Gradienten Θ_i am Punkt p_i

Die Verbindung zwischen den Punkten beschreibt man mit Geradenabschnitten, die in der Hesseschen Normalform aufgestellt werden. Für Geraden, die nicht durch den Nullpunkt gehen, lautet die Hessesche Normalform:

$$x \cos \beta + y \sin \beta = r \quad \text{für} \quad r \neq 0 \tag{5.16}$$

$$0 \leq \beta \leq 360\ Grad \tag{5.17}$$

$$r \leq \sqrt{xmax^2 + ymax^2} \tag{5.18}$$

Der Parameter r entspricht der Länge des Lotvektors vom Nullpunkt zur Geraden und der Winkel β beschreibt die Richtung der Geraden auf der Basis eines Einheitsvektors (siehe Abb. 5.7). Die Begrenzung der Definitionsbereiche sorgt für eine problemlose Quantisierung der Parameter.

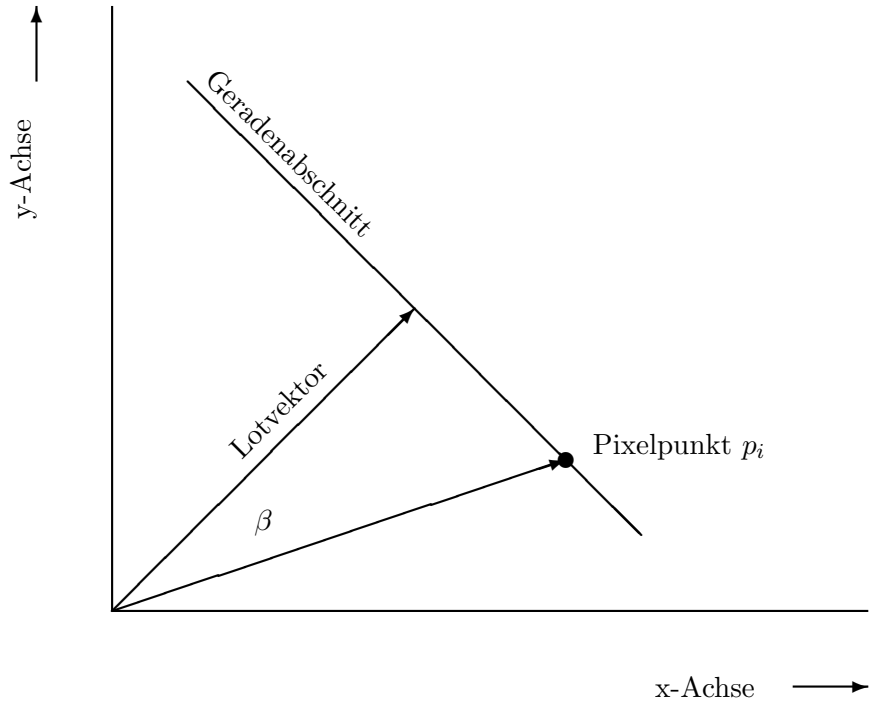


Abb.: 5.7: Hessesche Normalform einer Geraden durch den Pixelpunkt p

Die allgemeine Parameter-Transformation von Hough nutzt diese Darstellungen. Ausgehend von einem Mittelpunkt x_m, y_m, der im Innern des graphischen Objektes liegen muss, läßt sich für jeden einzelnen Punkt einer Randlinie ein Satz aus drei Parametern berechnen:

$$\text{Gradientenwinkel} \quad \theta_i = \arctan\left(\frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}}\right) \tag{5.19}$$

$$\text{Geradenabstand} \quad r = \sqrt{(x_i - x_m)^2 + (y_i - y_m)^2} \tag{5.20}$$

$$\text{Geradenwinkel} \quad \beta = \arctan\left(\frac{y_i - y_m}{x_i - x_m}\right) \tag{5.21}$$

Die Wertepaare (r, β) fasst man in einer Tabelle zusammen, die in der Literatur als R-Matrix bekannt ist und dessen Zeilenanzahl der Anzahl der betrachteten Gradientenwinkel entspricht. Die R-Matrix sieht dann für n Pixelpunkte wie folgt aus

$$R = \begin{pmatrix} \theta_1 & (r_{1,1}, \beta_{1,1}) & \cdots & (r_{1,m}, \beta_{1,m}) \\ \vdots & \vdots & \vdots & \vdots \\ \theta_n & (r_{n,1}, \beta_{n,1}) & \cdots & (r_{n,m}, \beta_{n,m}) \end{pmatrix} \tag{5.22}$$

(5.23)

$$RR = \begin{pmatrix} r_{1,1} & \cdots & r_{1,m} \\ \vdots & \vdots & \vdots \\ r_{n,1} & \cdots & r_{n,m} \end{pmatrix} \quad \text{bzw.} \quad RW = \begin{pmatrix} \beta_{1,1} & \cdots & \beta_{1,m} \\ \vdots & \vdots & \vdots \\ \beta_{n,1} & \cdots & \beta_{n,m} \end{pmatrix} \tag{5.24}$$

Jede Zeile entspricht einem Gradientenwinkel und jede Spalte nimmt ein Parameterpaar der Geraden vom Mittelpunkt zum Randpunkt auf. Falls die Randlinie an m Stellen den gleichen Gradienten aufweist, enthält die R-Matrix ebenfalls m Spalten mit Parameterpaaren.

Zur Erkennung einer Struktur geht man nun die einzelnen Pixelpunkte der Randlinie durch und bestimmt von jedem den Gradientenwinkel, der wiederum einer Zeile in den beiden R-Matrizen entspricht. Für jeden Spalteneintrag sind nun die Mittelpunktskoordinaten als Parameter zu berechnen und einem ganzzahligen Index der Akkumulator-Matrix zuzuordnen.

$$x_m = x - r\cos\beta \quad \longrightarrow \quad i = \frac{x_m}{x_{max}}\, imax \tag{5.25}$$

$$y_m = y - r\sin\beta \quad \longrightarrow \quad j = \frac{y_m}{y_{max}}\, jmax \tag{5.26}$$

$$A_{i,j} = A_{i,j} + 1 \tag{5.27}$$

Das maximale Element der A-Matrix liefert die gesuchten Parameter zurück.

Man kann dieses Verfahren noch um einen Skalierungsfaktor s und einen Rotationswinkel α erweitern. Dann lassen sich Objekte erkennen, die gegenüber der in der R-Matrix gespeicherten Struktur um s skaliert und um α gedreht sind.

$$x_m = x - s \cdot r \cdot \cos(\alpha + \beta) \quad \longrightarrow \quad i = \frac{x_m}{x_{max}}\, imax \tag{5.28}$$

$$y_m = y - s \cdot r \cdot \sin(\alpha + \beta) \quad \longrightarrow \quad j = \frac{y_m}{y_{max}}\, jmax \tag{5.29}$$

$$A_{i,j,v,w} = A_{i,j,v,w} + 1 \tag{5.30}$$

s und α werden wieder in den Bereichen schrittweise verändert. Die Akkumulator-Matrix umfaßt nun 4 Dimensionen, was den Rechenaufwand deutlich erhöht. Allerdings besitzt dieser erweiterte Algorithmus die erstaunliche Fähigkeit, ein mit einem unbekannten Faktor s multipliziertes und um den unbekannten Winkel α gedrehtes eingescanntes Pixelbild auf das gespeicherte Referenzbild zu transformieren.

$$s_{min} \le s \le s_{max} \quad \longrightarrow \quad 0 \le v \le v_{max} \tag{5.31}$$

$$\alpha_{min} \le \alpha \le \alpha_{max} \quad \longrightarrow \quad 0 \le w \le w_{max} \tag{5.32}$$

Bei $s \approx 1$ und $\alpha \approx 0$ reichen wenige Quantisierungsstufen aus und die in der Akkumulator-Matrix gespeicherte Datenmenge vom Typ byte nimmt nur leicht zu. Daher sollte man in diesem Fall die Gleichungen (5.31) und (5.32) auf jeden Fall verwenden.

Beispiel 5.3:

Die Abb. 5.8 stellt die in der Gleichung (5.24) definierten RR- bzw. die RW-Matrizen für zwei Testbilder dar. Beide Bilder bestehen aus 12 Punkten. Die Einteilung der Gradientenwinkel erfolgt ebenfalls in 12 Schritten, d.h. der Winkelbereich zwischen 0 Grad und 180 Grad quantisiert sich in Stufen von jeweils 15 Grad. Der Mittelpunkt beider Bilder liegt im Zentrum des weißen Graphikfensters.

Für das linke Bild des zentrierten Kreises folgen konstante Abstände $r = 60$ an den jeweiligen Winkelstufen, die jeweils vor der ersten Spalte als Kommentar angegeben sind. Jeder Eintrag in die RR-Matrix taucht doppelt auf, denn bei einem kreisförmigen Objekt sind die Beträge der Gradienten auf den gespiegelten Punkten identisch. Dies gilt sinngemäß auch für die Winkelangaben β in der RW-Matrix, denn hier addieren sich die Winkel der gespiegelten Punkte zu 360 Grad.

Im rechten Bild der Abb. 5.8 dient eine Ellipse als Testbild. Nun verändern sich die Radien und auch für die Winkel ergeben sich neue Werte. Die Summe der Winkelwerte addiert sich ebenfalls wieder zu 360 Grad auf.

RR-Matrix: 1

0	60	60
15		
30	60	60
45		
60	60	60
75		
90	60	60
105		
120	60	60
135		
150	60	60
165		

RW-Matrix: 1

0	179	359
15		
30	29	330
45		
60	60	299
75		
90	269	89
105		
120	240	119
135		
150	209	150
165		

RR-Matrix: 2

0	60	60
15	68	68
30		
45	83	83
60		
75		
90	90	90
105		
120		
135	83	83
150		
165	68	68

RW-Matrix: 2

0	179	359
15	40	319
30		
45	68	291
60		
75		
90	269	89
105		
120		
135	248	111
150		
165	220	139

Abb.: 5.8: Beispiele für die R-Matrizen der allgemeinen Hough-Transformation

Die Berechnung der R-Matrizen erfordert die Bestimmung von Gradienten und setzt damit die Verfügbarkeit von geordneten Punktfolgen auf einer Randline voraus. Aus der Differenz zwischen den jeweiligen x- bzw. y-Koordinaten des nachfolgenden Punktes p_{i+1} und dem zurück liegenden Punkt p_{i-1} lassen sich die Gradienten gemäß der Gleichung (5.19) bestimmen. Daher ist die allgemeine Hough-Transformation nur dann anwendbar, wenn eine entsprechende Randlinie des zu erkennenden graphischen Objektes vorhanden ist.

Falls ein graphisches Objekt durch eine Punktwolke gegeben ist, dann lassen sich durch geeignete Filterverfahren zur Kantenverstärkung die erforderlichen Randlinien gewinnen. Leider wirken sich unterschiedliche Beleuchtungsstärken sehr stark auf den Erfolg der Randlinienbestimmung aus. Daher sollte man die allgemeine Hough-Transformation nur dann einsetzen, wenn die zu erkennenden graphischen Objekte schon durch Randlinien definiert sind.

Beispiel 5.4:

Die Abb. 5.9 stellt in der linken oberen Bildhälfte die Ausgangssituation dar. Die schwarze größere Randlinie entspricht der gespeicherten Referenzgraphik. Die rote Randline zeigt das gleiche graphische Objekt, allerdings wie es von einer Kamera erfaßt wurde.

Im ersten Schritt erfolgt die Kompensation des verschobenen Mittelpunktes. Die Darstellung oben rechts in der Abb. 5.9 läßt erkennen, wie ausgehend vom roten gescannten Linienzug die Hough-Transformation den grünen Linienzug erzeugt. Die grauen Linien stellen den Verlauf der Verschiebung anschaulich in Form einer Animation dar.

In der linken unteren Abbildung erfolgt die Kompensation der Drehung von der roten Ausgangslinie, über die grauen Linien bis zum Ergebnis, das als grüne Linie dargestellt ist.

Schließlich fehlt noch die Kompensation des Skalierungsfaktors s. Der rote Linienzug, bei dem Mittelpunkt und Drehwinkel bereits kompensiert sind, geht nun in der rechten unteren Darstellung in den grünen Linienzug über.

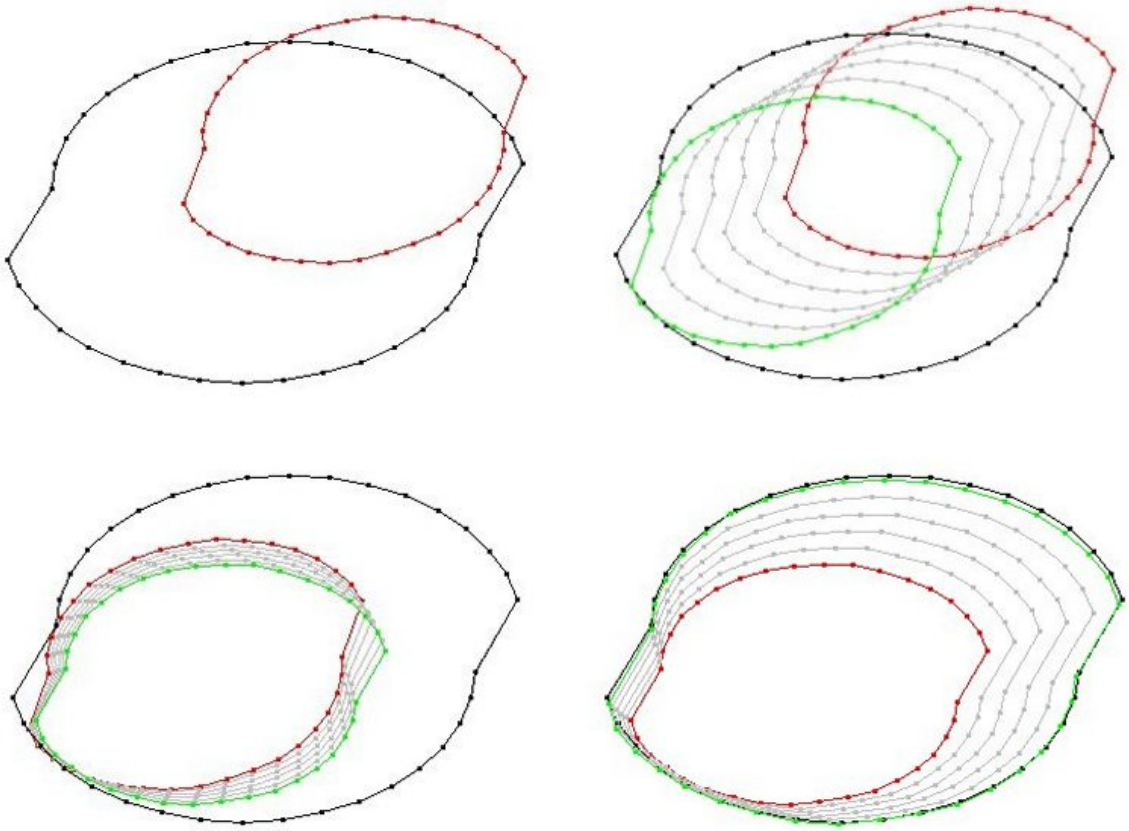


Abb.: 5.9: Beispiel einer allgemeinen Hough-Transformation

Die schrittweisen Übergänge von den roten zu den grünen Linienzügen, die in Form der grauen Linien visualisiert sind, dienten zur Veranschaulichung der allgemeinen Hough-Transformation. Der Kern-Algorithmus berechnet die Parameter

$$x_m, \qquad y_m, \qquad s \qquad \alpha$$

in einem Schritt. Die Rechenzeit liegt insgesamt unterhalb einer Sekunde. Damit erweist sich die allgemeine Hough-Transformation als gut geeignet zur Bilderkennung unter Echtzeit-Bedingungen.

5.4 Matching mittels mehrdimensionaler Regression

Regressionsverfahren gehen von einer in der Regel gemessenen Zahlenfolge aus, die ein analytischer Funktionsverlauf näherungsweise so verbindet („ausgleicht“), dass die Summe der Fehlerquadrate zwischen den gegebenen Meßwerten und den entsprechenden Funktionswerten ein Minimum annimmt. Wählt man als Funktion eine Gerade und als Meßdaten z.B. die einzelnen Werte eines bestimmten Aktienkurses, dann liefern die Parameter der Ausgleichsgeraden den Mittelwert und den Anstieg des Kursverlaufes für ein zeitliches Intervall.

Die elementare Methode der Regression eignet sich zur Kompensation von unerwünschten geometrischen Transformationen, die bei der Bilderkennung dafür verantwortlich sind, dass die gescannte Aufnahme eines Objekts erheblich von der gespeicherte Aufnahme des gleichen Objekts abweichen kann.

Angenommen, die gescannte Aufnahme ist gegenüber dem Referenzbild um Δx und um Δy verschoben, um s_x bzw. s_y skaliert und um den Winkel α gedreht, dann entspricht dies einer Multiplikation der Pixel-Koordinaten der Referenzaufnahme mit der Transformationsmatrix P:

$$P = \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.33}$$

$$= \begin{pmatrix} s_x\cos\alpha & -s_x\sin\alpha & \Delta x \\ s_y\sin\alpha & s_y\cos\alpha & \Delta y \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} p_{1,1} & p_{1,2} & p_{1,3} \\ p_{2,1} & p_{2,2} & p_{2,3} \\ 0 & 0 & 1 \end{pmatrix} \tag{5.34}$$

Wären die Elemente der Transformationsmatrix P bekannt, dann könnte man die Pixelkoordinaten der Referenzbilder r_i eines graphischen Objekts, z.B. einer Randlinie, mit der P-Matrix transformieren und mit den korrespondierenden gescannten Pixel-Koordinaten z_i zur Übereinstimmung bringen („matchen“).

$$\begin{pmatrix} p_{1,1} & p_{1,2} & p_{1,3} \\ p_{2,1} & p_{2,2} & p_{2,3} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} r_{1,1} & \cdots & r_{1,i} & \cdots & r_{1,m} \\ r_{2,1} & \cdots & r_{2,i} & \cdots & r_{2,m} \\ 1 & \cdots & 1 & \cdots & 1 \end{pmatrix} = \begin{pmatrix} z_{1,1} & \cdots & z_{1,i} & \cdots & z_{1,m} \\ z_{2,1} & \cdots & z_{2,i} & \cdots & z_{2,m} \\ 1 & \cdots & 1 & \cdots & 1 \end{pmatrix} \tag{5.35}$$

Die Matrizen R und Z bestehen aus drei Zeilen für homogene zwei-dimensionale Koordinaten und m Spalten, wobei m der Anzahl der zu transformierenden Bildpunkte entspricht. Zur Begrenzung des Rechenaufwandes eignet sich ein Schwellwert, der nur jene Pixel auswählt, die einen hohen Informationsgehalt besitzen.

Die Gleichung (5.35) besteht aus drei Matrizen, wobei die linke Matrix unbekannt ist. Daher ist diese Matrizengleichung lösbar. Das Lösungsverfahren startet mit der Transponierung der Gleichung (5.35). Gemäß den Grundregeln der linearen Algebra kehrt sich beim Transponieren eines Matrizenproduktes die Reihenfolge der Multiplikation um.

$$\begin{pmatrix} r_{1,1} & r_{2,1} & 1 \\ \vdots & \vdots & \vdots \\ r_{1,i} & r_{2,i} & 1 \\ \vdots & \vdots & \vdots \\ r_{1,m} & r_{2,m} & 1 \end{pmatrix} \cdot \begin{pmatrix} p_{1,1} & p_{2,1} & 0 \\ p_{1,2} & p_{2,2} & 0 \\ p_{1,3} & p_{2,3} & 1 \end{pmatrix} = \begin{pmatrix} z_{1,1} & z_{2,1} & 1 \\ \vdots & \vdots & \vdots \\ z_{1,i} & z_{2,i} & 1 \\ \vdots & \vdots & \vdots \\ z_{1,m} & z_{2,m} & 1 \end{pmatrix} \tag{5.36}$$

Die linksseitige Multiplikation der Gleichung (5.36) mit der ursprünglichen R-Matrix liefert das in der Gleichung (5.37) dargestellte Zwischenergebnis:

$$\begin{pmatrix} r_{1,1} & \cdots & r_{1,i} & \cdots & r_{1,m} \\ r_{2,1} & \cdots & r_{2,i} & \cdots & r_{2,m} \\ 1 & \cdots & 1 & \cdots & 1 \end{pmatrix} \cdot \begin{pmatrix} r_{1,1} & r_{2,1} & 1 \\ \vdots & \vdots & \vdots \\ r_{1,i} & r_{2,i} & 1 \\ \vdots & \vdots & \vdots \\ r_{1,m} & r_{2,m} & 1 \end{pmatrix} \cdot \begin{pmatrix} p_{1,1} & p_{2,1} & 0 \\ p_{1,2} & p_{2,2} & 0 \\ p_{1,3} & p_{2,3} & 1 \end{pmatrix} \tag{5.37}$$

$$= \begin{pmatrix} r_{1,1} & \cdots & r_{1,i} & \cdots & r_{1,m} \\ r_{2,1} & \cdots & r_{2,i} & \cdots & r_{2,m} \\ 1 & \cdots & 1 & \cdots & 1 \end{pmatrix} \cdot \begin{pmatrix} z_{1,1} & z_{2,1} & 1 \\ \vdots & \vdots & \vdots \\ z_{1,i} & z_{2,i} & 1 \\ \vdots & \vdots & \vdots \\ z_{1,m} & z_{2,m} & 1 \end{pmatrix} \tag{5.38}$$

Durch die linksseitige Multiplikation mit der R-Matrix entsteht das Produkt der R-Matrix mit seiner transponierten Matrix. Dieses spezielle Matrizenprodukt ergibt eine quadratische Matrix Q, die nur noch aus drei Zeilen bzw. Spalten besteht.

$$Q = \begin{pmatrix} r_{1,1} & \cdots & r_{1,i} & \cdots & r_{1,m} \\ r_{2,1} & \cdots & r_{2,i} & \cdots & r_{2,m} \\ 1 & \cdots & 1 & \cdots & 1 \end{pmatrix} \cdot \begin{pmatrix} r_{1,1} & r_{2,1} & 1 \\ \vdots & \vdots & \vdots \\ r_{1,i} & r_{2,i} & 1 \\ \vdots & \vdots & \vdots \\ r_{1,m} & r_{2,m} & 1 \end{pmatrix} \tag{5.39}$$

Mit der Q-Matrix sieht die Gleichung (5.39) wie folgt aus:

$$Q \cdot \begin{pmatrix} p_{1,1} & p_{2,1} & 0 \\ p_{1,2} & p_{2,2} & 0 \\ p_{1,3} & p_{2,3} & 1 \end{pmatrix} = \begin{pmatrix} r_{1,1} & \cdots & r_{1,i} & \cdots & r_{1,m} \\ r_{2,1} & \cdots & r_{2,i} & \cdots & r_{2,m} \\ 1 & \cdots & 1 & \cdots & 1 \end{pmatrix} \cdot \begin{pmatrix} z_{1,1} & z_{2,1} & 1 \\ \vdots & \vdots & \vdots \\ z_{1,i} & z_{2,i} & 1 \\ \vdots & \vdots & \vdots \\ z_{1,m} & z_{2,m} & 1 \end{pmatrix} \tag{5.40}$$

Als letzten Schritt berechnet man die inverse Q-Matrix und führt eine linksseitige Multiplikation der Gleichung (5.37) durch:

$$Q^{-1} \cdot Q \cdot \begin{pmatrix} p_{1,1} & p_{2,1} & 0 \\ p_{1,2} & p_{2,2} & 0 \\ p_{1,3} & p_{2,3} & 1 \end{pmatrix} = Q^{-1} \cdot \begin{pmatrix} r_{1,1} & \cdots & r_{1,i} & \cdots & r_{1,m} \\ r_{2,1} & \cdots & r_{2,i} & \cdots & r_{2,m} \\ 1 & \cdots & 1 & \cdots & 1 \end{pmatrix} \cdot \begin{pmatrix} z_{1,1} & z_{2,1} & 1 \\ \vdots & \vdots & \vdots \\ z_{1,i} & z_{2,i} & 1 \\ \vdots & \vdots & \vdots \\ z_{1,m} & z_{2,m} & 1 \end{pmatrix} \tag{5.41}$$

Das Produkt einer inversen Matrix mit der Ausgangsmatrix führt zur Einheitsmatrix, d.h. die Gleichung (5.42) beginnt nun mit der gesuchten P-Matrix:

$$\begin{pmatrix} p_{1,1} & p_{2,1} & 0 \\ p_{1,2} & p_{2,2} & 0 \\ p_{1,3} & p_{2,3} & 1 \end{pmatrix} = Q^{-1} \cdot \begin{pmatrix} r_{1,1} & \cdots & r_{1,i} & \cdots & r_{1,m} \\ r_{2,1} & \cdots & r_{2,i} & \cdots & r_{2,m} \\ 1 & \cdots & 1 & \cdots & 1 \end{pmatrix} \cdot \begin{pmatrix} z_{1,1} & z_{2,1} & 1 \\ \vdots & \vdots & \vdots \\ z_{1,i} & z_{2,i} & 1 \\ \vdots & \vdots & \vdots \\ z_{1,m} & z_{2,m} & 1 \end{pmatrix} \tag{5.42}$$

Zusammenfassend ergibt sich das folgende Gleichungssystem:

$$T = (R^T R)^{-1} R \tag{5.43}$$

$$(T \cdot Z_{Scan}) \cdot R = Z_{Referenz} \tag{5.44}$$

$$s_i = \sqrt{\frac{1}{m} \sum_{i=1}^{i=m} (Z_{Scan}(i,1) - Z_{Ref}(i,1))^2 + (Z_{Scan}(i,2) - Z_{Ref}(i,2))^2} \tag{5.45}$$

Die Auswertung der Lösungsgleichungen (5.43), (5.44) und (5.45) erfordert mehrere Schritte und ist in der Abb. 5.10 graphisch veranschaulicht:

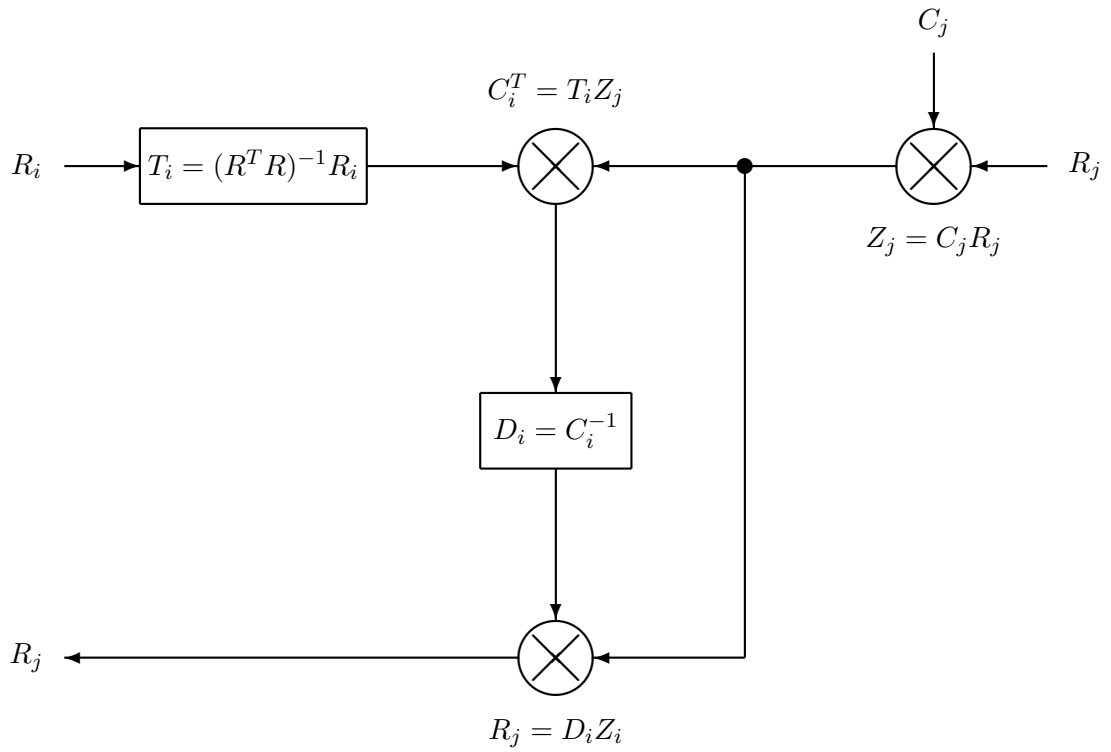


Abb.: 5.10: Regression von Pixeldaten

Beim Programmstart erfolgt für jedes Referenzbild R_i die Berechnung der dazu gehörenden T-Matrix T_i. Eine unverfälschte gescannte Aufnahme R_j, die weitgehend mit einem der Referenzbilder R_i übereinstimmt und durch die Kameraposition einer unbekannten Transformation C_j unterliegt, wird mit der Bezeichnung Z_j mit den vorab berechneten Matrizen T_i multipliziert. Als Ergebnis dieser Multiplikation entsteht die transponierte Transformationsmatrix C_i, welche den Übergang eines Referenzbildes R_i auf die gescannte Aufnahme R_j beschreibt.

Der Bildvergleich erfolgt auf der Ebene der Referenzbilder R_i. Deshalb ist die Transformationsmatrix C_i zu invertieren und mit der Bezeichnung D_i auf den Bildscan Z_j anzuwenden. Jenes Referenzbild R_i, das mit dem auf Referenzebene transformierten Bildscan R_j am besten übereinstimmt, gilt dann als erkannt. Als Maß der Übereinstimmung läßt sich die skalare Quadratsumme d aller n Elemente der Differenz der Matrizen R_i und R_j bestimmen. Das Minimum von d_i stellt sich beim Index $imin$ ein. n bezeichnet die Anzahl der Pixelpunkte, die jeweils mit ihren x- bzw. y-Koordinaten auftreten. Die dritte Komponente des Pixelvektors entspricht der Eins homogener Koordinaten.

$$d_i = \sum_{k=1}^{k=n} \left[(\ R_i(k,1) - R_j(k,1)\ )^2 + (\ R_i(k,2) - R_j(k,2)\ )^2 \right] \tag{5.46}$$

$$R_j = R_{imin} \tag{5.47}$$

Beispiel 5.4:

Die Abb. 5.11 stellt einen Ausschnitt aus einer Animation dar. Die schwarzen Graphiken entsprechen berechneten und zufällig variierten Referenzbildern. Dann wird eines der Referenzbilder ebenfalls zufällig ausgewählt, mit einer zufallsgesteuerten geometrischen Transformation verändert und in der Abb. 5.11 unten am rechten Rand als grüner Linienzug dargestellt. Der menschliche Betrachter ist nicht in der Lage, das passende Referenzbild aus den dargestellten Bildern zu erkennen. Da es sich hier um eine Animation handelt, sind die in der praktischen Anwendung immer unbekannten Transformationsparameter sogar bekannt. In der Abb. 5.11 wurde eines der Referenzbilder in der x-Achse um den Faktor 1,3 gestreckt, in der y-Achse um den Faktor 0.7 gestaucht, um 20 Grad im Uhrzeigersinn gedreht und um jeweils 10 Pixelpunkte verschoben. Doch welches der Referenzbilder könnte das sein ? Trotz dieser zusätzlichen Angaben läßt sich die Lösung kaum erraten.

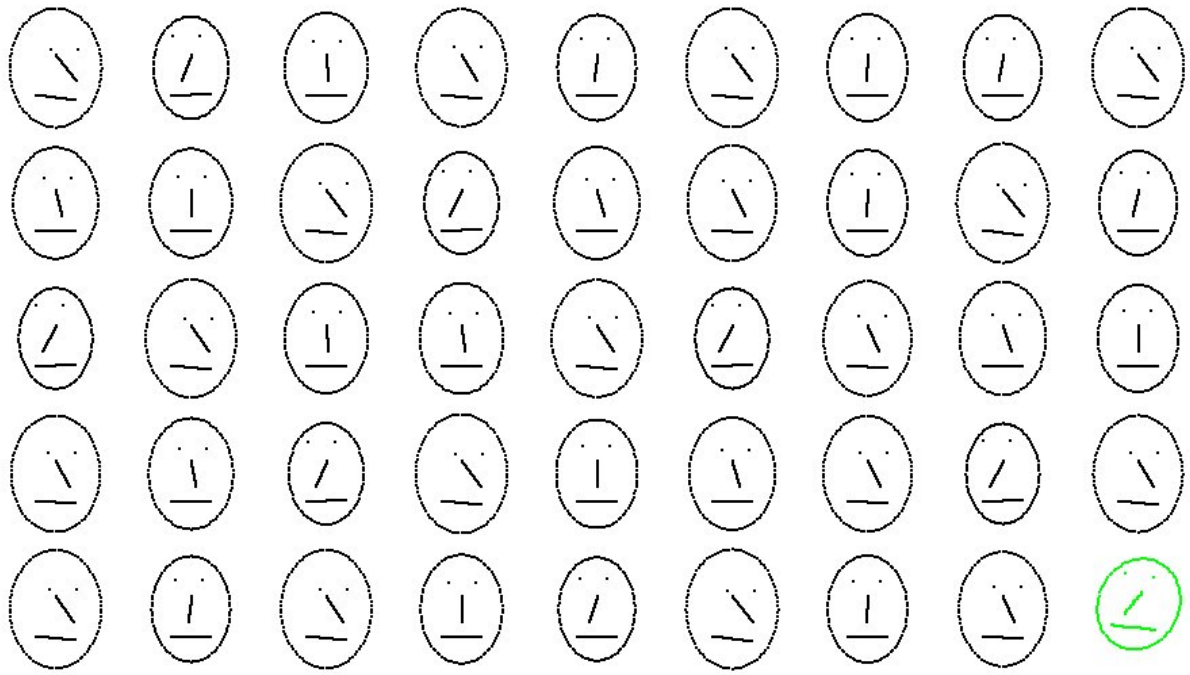


Abb.: 5.11: Welches schwarze Bild stimmt mit dem grünen Bild überein ?

Die Algorithmen der Regression lösen diese Aufgabe problemlos in weniger als einer Sekunde. Das gefundene Gesicht wird durch die grüne Farbe markiert. Das in die Ebene der Referenzbilder transformierte grüne Ausgangsbild ist in roter Farbe über die Referenzbilder gezeichnet. Nur das gesuchte Bild weist eine fehlerfreie Übereinstimmung auf. Die Fehlerwerte aller Bilder kann man in der Säulendarstellung am unteren Bildrand der Abb. 5.12 erkennen.

Abb.: 5.12: Lösung der Bildsuche

Diese Animation kann beliebig lange ablaufen. Bei einer richtigen Lösung erhöht sich ein entsprechender Zählerstand, der durch die Anzahl der automatischen Testläufe geteilt und als prozentuale Erkennungsrate als Säule rechts neben den Säulendarstellungen der Einzelfehler in der Abb. 5.12 zu erkennen ist. Da in dieser Animation kein Rauschen auftritt und alle Variationen auf linearen Transformationen basieren, beträgt die Erkennungsrate exakt 100%.

Beispiel 5.5:

Das Beispiel 5.5 entspricht weitgehend dem Beispiel 5.4, jedoch basieren die Bilder nun auf wirklichen Kamerabildern. Die Abb. 5.13 stellt in der obersten Reihe zehn beliebige Referenzaufnahmen im Format 80x80 Pixel dar. Das elfte Bild zeigt die gescannte Aufnahme, die in diesem Fall mit dem ersten Referenzbild identisch ist. In der zweiten Bildreihe darunter erkennt man die ausgewählten Pixelpunkte, die als Grundlage für das Regressionsverfahren dienen. Ein Pixel eignet sich dann für die Regression, wenn sein Grauwert unterhalb des Mittelwerts aller Grauwerte liegt und wenn der Abstand zum davor liegenden Grauwert mindestens 10 Pixelpunkte beträgt. Die grünen Pixelpunkte des Bildes am rechten Rand der zweiten Reihe stellen die gescannte Aufnahme dar.

Die dritte Bildreihe von oben enthält nun die in die Ebene der Referenzbilder transformierten Punkte des Bildscans. Da bei der Abb. 5.13 als Bildscan das erste Referenzbild ausgewählt wurde, stimmen im ersten Bild die Positionen aller Pixelpunkte exakt überein und der entsprechende Fehlerwert der Säulendarstellung nimmt den Wert Null an. Der nächste größere Fehlerwert liegt erst bei 55.

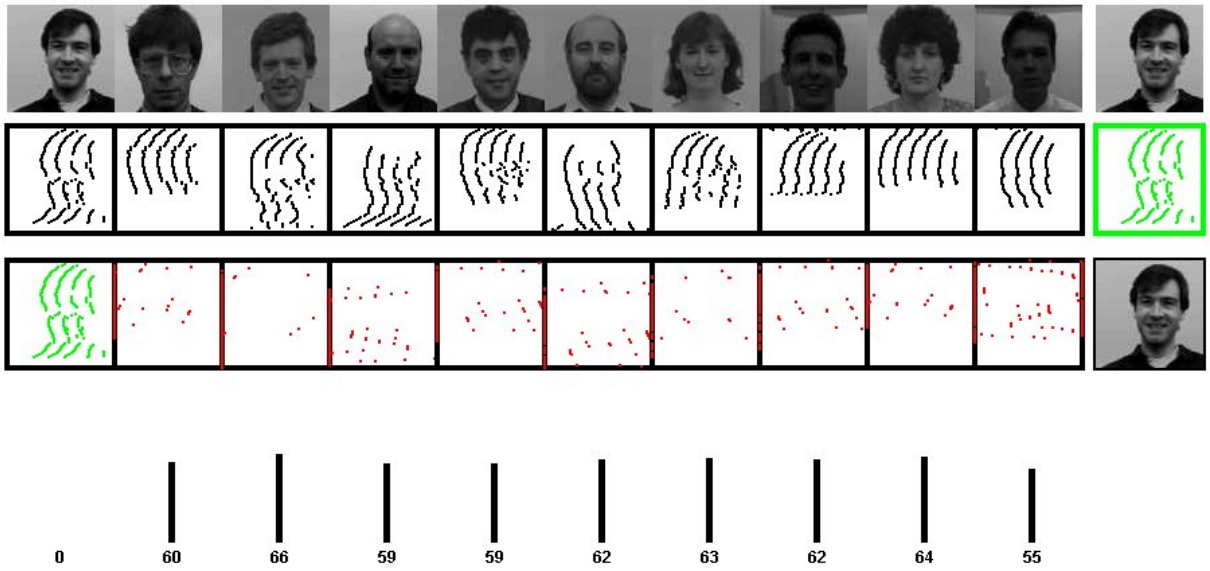


Abb.: 5.13: Matching bei identischen Bildern

Die Stärke eines Regressionsverfahrens liegt in seiner Fähigkeit mit verrauschten Datensätzen gut umgehen zu können. Diese hervorragende Eigenschaft demonstriert die Abb. 5.14, bei der den Pixel-Koordinaten des ersten Referenzbildes ein Rauschanteil von 50% überlagert ist. Obwohl das menschliche Auge im Pixelbild nichts mehr erkennen kann, liefert das Regressionsverfahren die richtige Person mit einem Fehler von 46% zurück. Der nächste Fehlerwert folgt erst bei 65%.

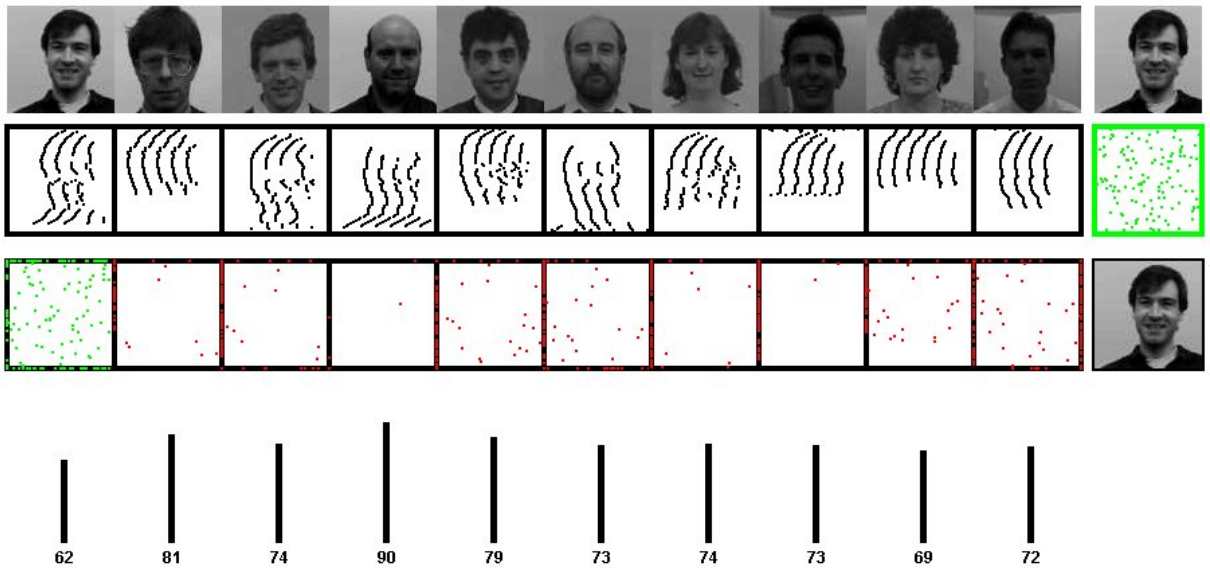


Abb.: 5.14: Matching bei verrauschten Bildern

Die Schwächen eines Regressionsverfahren treten dann hervor, wenn das Referenzbild auch inhaltlich vom gescannten Bild abweicht. In der Abb. 5.15 hält die erste Person beim Scannen die Hand zum Gruß vor die Stirn. Ohne Rauschanteil erkennt der Algorithmus die richtige Person, wobei der Fehler 59% beträgt und dicht am nächsten größeren Fehlerwert von 60% liegt.

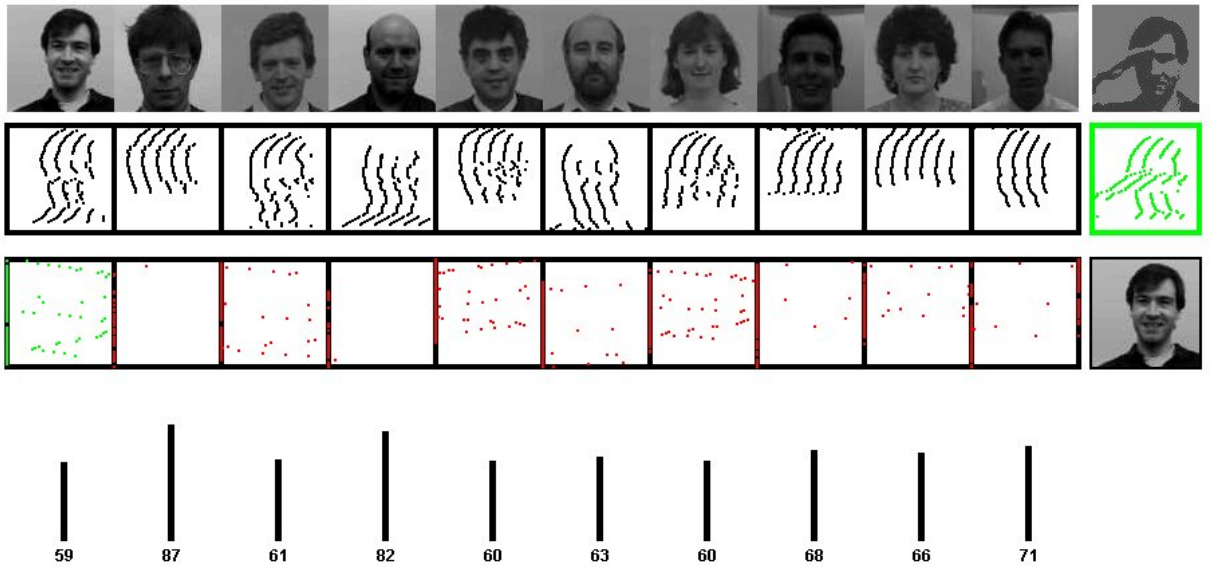


Abb.: 5.15: Matching bei unterschiedlichen Bildinhalten ohne Rauschen

Ein ähnliches Ergebnis folgt auch für die „Frau mit Brille“ (Abb. 5.16), die bei einem Rauschanteil von 50% gerade noch erkannt werden kann. Allerdings sollten Ergebnisse, die mit einem so hohen Fehleranteil wie z.B. mit dem hier vorliegenden Wert von 72% behaftet sind, nicht mehr als Lösung akzeptiert werden.

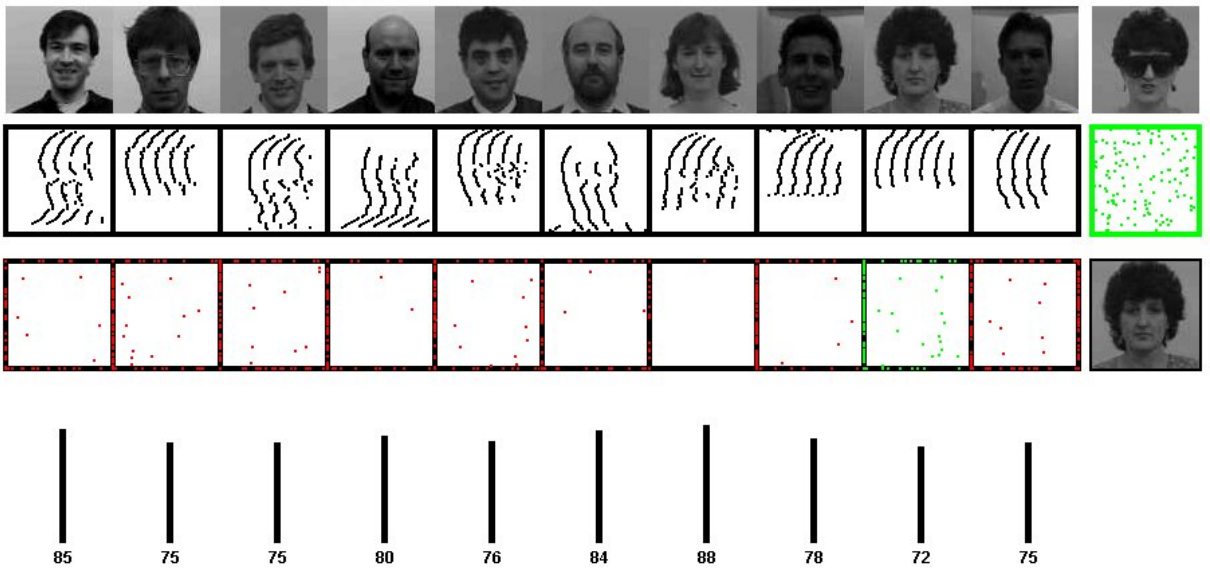


Abb.: 5.16: Matching bei unterschiedlichen Bildinhalten mit Rauschen

Kapitel 6

Der Aufbau des Erkennungssystems

6.1 Die Module des Erkennungssystems

Bei der Erkennung eines Bildes geht es darum, das Bild einer Digitalkamera, ein Webcam-Bild oder ein Scanner-Bild auszuwerten. Automatisches Navigieren sowie alle Arten von Objekterkennung fallen in dieses Fachgebiet. Von besonders großem wirtschaftlichem und sicherheitspolitischem Interesse ist die automatische Personenerkennung auf öffentlichen Plätzen durch die schnelle Auswertung von Überwachungskameras. Menschen sind in der Regel überfordert, wenn sie für längere Zeit eine derartige Überwachung durchführen sollen.

Die automatische Personenerkennung stellt hohe Anforderungen an die Leistungsfähigkeit der eingesetzten Algorithmen. Eine geeignete Rechnerarchitektur ermöglicht die erforderliche Echtzeitfähigkeit dieses Systems. Es kommt darauf an, die bisher vorgestellten algorithmischen Verfahren für ein Erkennungssystem auf optimale Weise zu kombinieren. Bezüglich der Eigenschaften der einzelnen Algorithmen gilt die Zielvorgabe:

Stärken sollen sich addieren und Schwächen sollen sich kompensieren.

Die Abb. 6.1 stellt die einzelnen Module des Systems zur Personenerkennung dar.

Multiprozessor-Board	⟶	Verteilung der Rechenlast auf mehrere Prozessoren
Wavelet - Kompression	⟶	Reduktion der Datenmenge
Digitale Filter	⟶	Normierung der Bildscans
Eigenwert - Methode	⟶	Erkennen, ob es sich wirklich um eine Person handelt
Neuronales Netz	⟶	Herausrechnen von Bildobjekten
Matching (Regression)	⟶	Intelligentes Erkennen teilweise vermummter Personen
Personen - Update	⟶	Update der Eigengesichter und der neuronalen Gewichte
Kamerasteuerung	⟶	Dynamische Kamerasteuerung mit Personen-Tracking
GUI	⟶	Graphische Oberfläche zur Bedienung
Software-Design	⟶	Drei-Schichten-Architektur

Abb.: 6.1: Die Module eines Gesichtserkennungssystems

Multiprozessor-Board
Die Algorithmen zur Erfassung und Speicherung von Referenzbildern einerseits und die echtzeit-orientierte Personenerkennung andererseits sollten jeweils auf einem eigenen Prozessor laufen, damit der Erkennungsprozess mit maximaler Geschwindigkeit erfolgen kann. Die Echtzeitforderung gilt nur für den eigentlichen Erkennungsprozess. Die Entwicklung und der Test der Algorithmen erfolgte auf Rechnern mit einem Dual-Prozessor Board auf der Basis eines shared-memory Systems und der openMP - Programmiertechnik unter Java.

Wavelet - Kompression
Die Originalbilder einer Überwachungskamera weisen in der Regel eine Anzahl von Pixeln auf, die für die nachfolgenden Algorithmen viel zu groß sind. Durch die Anwendung der diskreten Wavelet-Transformation gelingt es, z.B. eine Auflösung von 640x480 Pixeln in drei Stufen auf eine Auflösung von 80x60 Pixeln zu reduzieren. Nach jeder Stufe verkleinert sich der Datenumfang auf ein Viertel und am Ende beträgt das Verhältnis der Datenreduktion 1:64. Der Informationsgehalt der reduzierten Bilder reicht für die Erkennungsverfahren aus.

$$\begin{array}{llllllll} 640 \times 480 \text{ Pixel} & \rightarrow & 320 \times 240 \text{ Pixel} & \rightarrow & 160 \times 120 \text{ Pixel} & \rightarrow & 80 \times 60 \text{ Pixel} \\ 307.200 \text{ Pixel} & \rightarrow & 77.800 \text{ Pixel} & \rightarrow & 19200 \text{ Pixel} & \rightarrow & 4800 \text{ Pixel} \end{array}$$

Digitale Filter
Digitale Filter sorgen für vergleichbare Bilder und unterstützen damit die Erkennungsalgorithmen. Das vorliegende System setzt die folgenden Filterfunktionen ein:

1. Mittelwertkompensation
2. Grauwertspreizung zur Nutzung des vollen Wertebereiches zwischen 0 und 255.
3. Zentrierung des Bildinhaltes mittels einer Schwerpunktberechnung
4. Laplace-Filter zur Kantenverstärkung unterstützen die nachfolgende Berechnung der Bildpunkte

Die Abb. 6.2 zeigt das Orginalbild, das aus 640x480 Pixeln besteht und im Vergleich dazu das links oben eingezeichnete reduzierte, gefilterte und zentrierte 80x60 Pixelbild, das als Eingangsbild für die Erkennungsalgorithmen dient.

Eigenwert - Methode
Die Eigenwert-Methode liefert einen prozentualen Zahlenwert für die Übereinstimmung zwischen der gescannten und den gespeicherten Aufnahmen. Damit läßt sich dann feststellen, ob ein gescanntes graphisches Objekt überhaupt ein Gesicht darstellt. Falls der relative Fehlerwert für das zunächst als erkannt angenommene gescannte Bild einen Grenzwert überschreitet, bricht die Erkennung ab und eine entsprechende Fehlermeldung erscheint. Somit lässt sich verhindern, dass eine „Teekanne“ fälschlich als Gesicht - wenn auch mit einem zu großen Fehlerwert - interpretiert wird.

Neuronales Netz
Das neuronale Netz bildet den eigentlichen Kern des Erkennungssystems. Selbst bei vermummten Personen ist es in der Lage, die verborgenen Gesichtsteile zu generieren. Allerdings verschlingt die iterative Auswertung der Recallphase die meiste Rechenzeit und steht im Mittelpunkt bei der Parallelisierung der Algorithmen.

Matching und Regression
Die Matching-Methode berechnet signifikante Bildpunkte. Eine geometrische Transformationsmatrix mit homogenen Koordinaten bildet die Punkte der gescannten Aufnahme

so auf alle der gespeicherten Aufnahmen ab, dass der Hausdorff-Abstand für eine der gespeicherten Aufnahmen ein Minimum annimmt. Der Einsatz des Hausdorff-Abstandes an Stelle der sonst üblichen minimalen Fehlerquadrate erhöht die Leistungsfähigkeit des Matching-Verfahrens erheblich, denn die Reihenfolge der Punkte spielt dabei keine Rolle mehr.

Personen - Update
Das System ist in der Lage weitere Personen in die Sammlung der gespeicherten Bilder aufzunehmen bzw. einzeln ausgewählte Bilder zu löschen. Diese „Lernfähigkeit“ des Erkennungssystems ermöglicht es während des Erkennungsbetriebes, neue Personen oder Personen in unterschiedlicher Aufnahmetechnik in die Bilddatenbank aufzunehmen und umgekehrt, beliebige Aufnahmen wieder zu löschen. Jede Änderung der Basisdaten zieht eine Neuberechnung der Start-Algorithmen nach sich.

Kamerasteuerung
Zur Zentrierung graphischer Objekte berechnet man die Mittelpunktskoordinaten. Das Ergebnis eignet sich dazu, die Überwachungskamera so zu steuern, dass sich die Person immer im zentralen Erfassungsbereich der Kamera aufhält. Damit lässt sich die Erkennung von Personen erheblich verbessern. Auch in diesem Zusammenhang ist es Bedingung, dass die Erkennungsalgorithmen schnell ablaufen, denn sonst wäre eine dynamische Kamera-Nachführung nicht möglich.

GUI
Die Entwicklung einer hilfreichen GUI ist für das Testen eines Erkennungssystem besonders wichtig. Neben der eigentlichen Anzeige der Ergebnisse erhält der Entwickler wertvolle Hinweise über den Betriebszustand des Systems und die Entscheidungen des Erkennungssystems lassen sich anschaulich nachvollziehen. Gerade in der Erprobungsphase eines Erkennungssystems ist es wichtig, die Arbeitsweise des Systems genau zu verfolgen und geeignete Kalibrierungen z.B. der Filterparameter vornehmen zu können.

Software-Design
Die Software des Erkennungssystems ist relativ umfangreich und gliedert sich in die persistenten Daten der Bildersammlung, der Algorithmen und der graphischen Darstellung der Ergebnisse. Daher bietet es sich an, die Softwareentwicklung in Form einer Drei-Schichten Architektur durchzuführen.

Abb.: 6.2: Reduktion, Filterung und Zentrierung eines Testbildes

6.2 Die logische Verknüpfung der Module

Die Abb. 6.3 stellt die logische Verknüpfung der Module beim Start des Programmes dar. Die Hauptaufgabe beim Start besteht darin, alle dann schon möglichen Berechnungen durchzuführen um die echtzeitkritischen Algorithmen zu entlasten.

Die Sammlung der gespeicherten Bilder gliedert man in Form einer Matrixstruktur mit M Zeilen und P Spalten. Für jede Zeile erfolgen getrennte Berechnungen der Eigenvektoren, der Bestimmung der Gewichte eines neuronalen Hopfield Netzes und der Generierung signifikanter Bildpunkte. Diese Begrenzung auf 5 Spalten ist vor allem aus numerischen Gründen erforderlich, denn nimmt man deutlich mehr Bilder in einer Zeile auf, dann steigt die Anzahl der Eigenvektoren und der zu speichernden Muster im Hopfield-Netz entsprechend an und die Leistungsfähigkeit dieser Verfahren würde rasch absinken. Die Aufteilung der Bilder in Zeilen und Spalten entspricht einer Parallelisierung der Algorithmen und unterstützt zusätzlich eine Implementierung auf parallelen Rechnerarchitekturen.

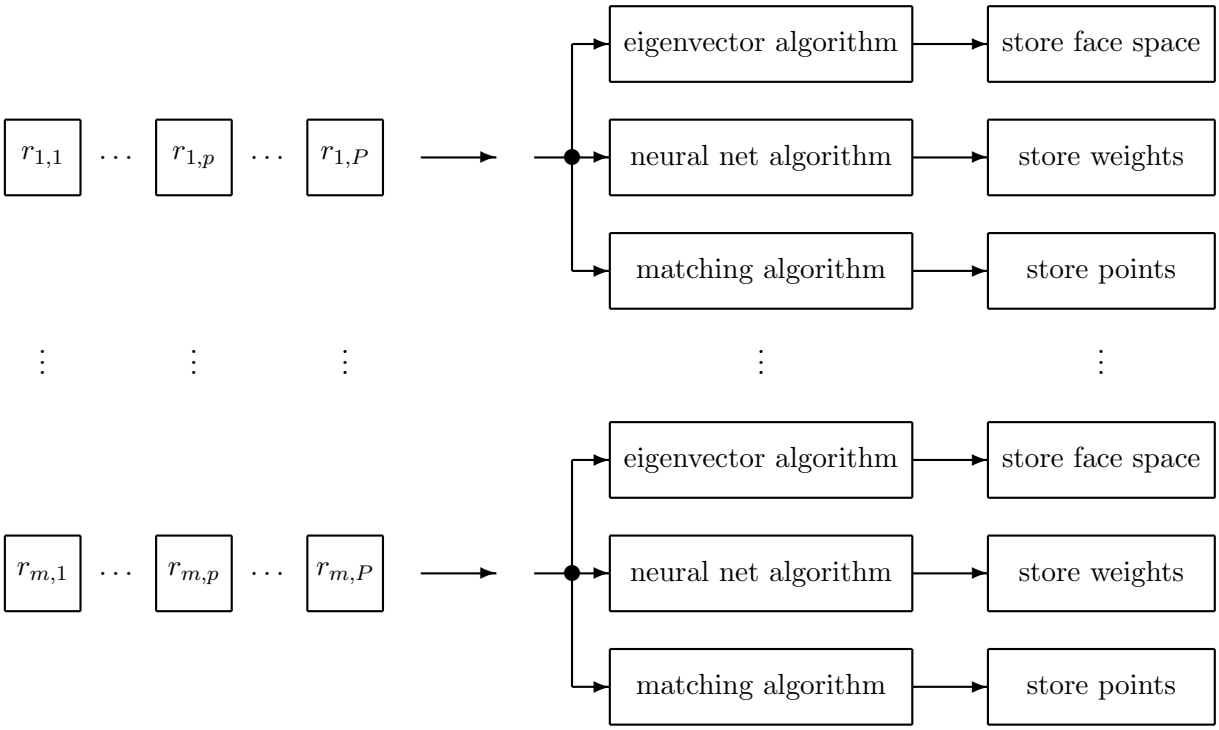


Abb.: 6.3: Die Module beim Start des Programms

Nachteilig an der in der Abb. 6.3 dargestellten Struktur ist allerdings der erhöhte Speicherbedarf, der dann beim Hopfield-Netz auftritt. Die Anzahl der Gewichte eines neuronalen Netzes hängt grundsätzlich nur von der Anzahl der Pixel N pro Bild ab und bei einer Aufteilung der $M \times P$ Bilder auf M neuronale Netze steigt die Anzahl n der Gewichte mit M linear an. Bei z.B. 15 Bildern mit jeweils 80x60 Pixeln bedeutet dies für eine 4 Byte Float-Darstellung der im Hauptspeicher des Rechners abzulegenden Gewichte:

$$n = N \cdot N \cdot M = (80 \cdot 60)^2 \cdot M = 23,04 \cdot 10^6 \cdot M \quad (6.1)$$

$$\rightarrow 23,04 \cdot 10^6 \cdot 4\ Byte \cdot M = 92,16\ \text{MByte} \cdot M = 460,8\ \text{MByte} \quad (6.2)$$

Durch eine numerische Umgestaltung der Algorithmen des Hopfield-Netzes gelang es, die Gewichte statt mit dem Typ „float“ mit dem Typ „byte“ zu speichern, was gemäß der Gleichung (6.2) den Speicherbedarf um ein Viertel auf 115, 2 MByte reduziert.

Der echtzeitfähige Erkennungsprozess wertet das gescannte Bild auf allen M parallelen Rechenzweigen einzeln aus. Wie in der Abb. 6.4 dargestellt, dienen die Ausgangsbilder der M Hopfield-Netze als Eingangsbild für die Generierung der signifikanten Bildpunkte des Matching-Verfahrens. Die Entscheidung, welches der gespeicherten Bilder als erkannt zu werten ist, erfolgt auf der Basis eines gewichteten und logischen Auswahlkriteriums. Die Eigenwertberechnungen, die Hopfield-Netze und die Matching-Verfahren liefern für die Abweichung des gescannten Bildes von den gespeicherten Bildern je einen Fehlerwert.

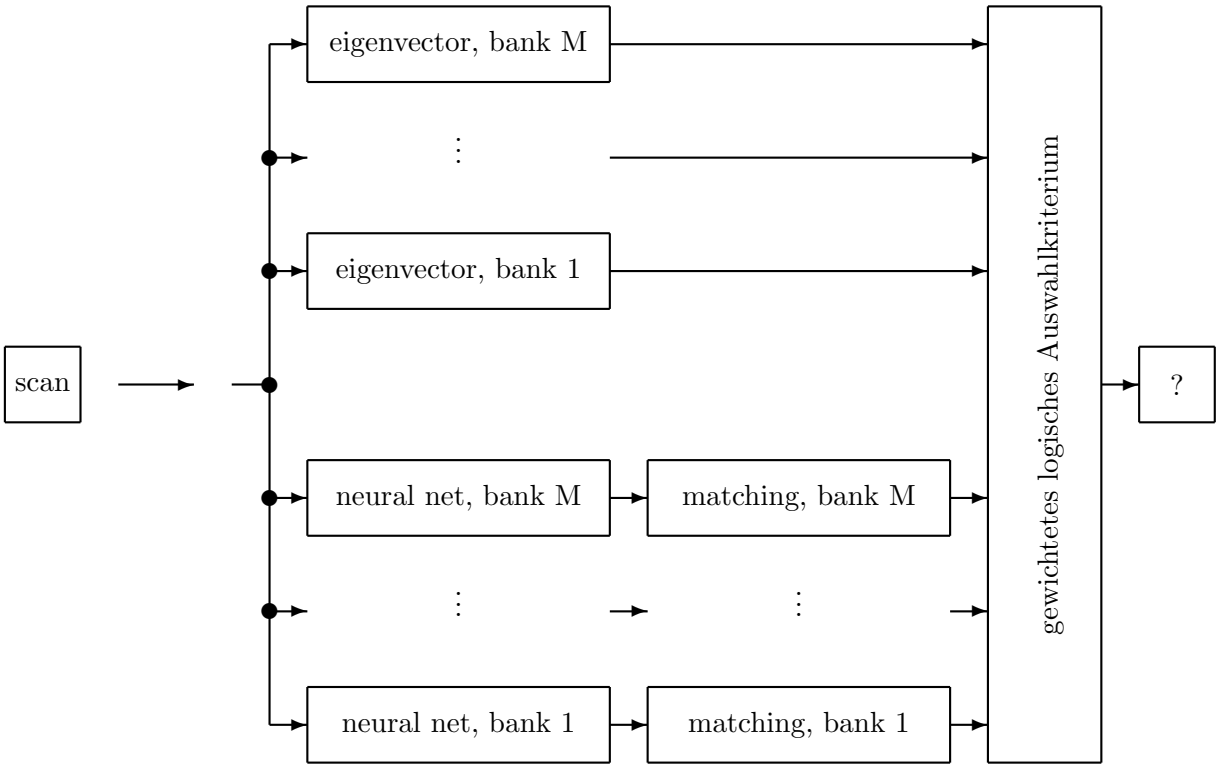


Abb.: 6.4: Die Module beim Erkennungsprozess des Programms

Insgesamt werden für jedes der gespeicherten Bilder drei Fehlerwerte der Übereinstimmung zum gescannten Bild berechnet. Falls einer der Fehlerwerte der Eigenwertberechnungen die Marke von 100% überschreitet, handelt es sich gemäß der Eigenwert-Methode um kein Gesicht. Die Berechnung des Gesamtfehlers erfolgt nach der Gleichung (6.3), wobei sich der kleinste Fehler immer durchsetzt und der Gesamtfehler unterhalb des kleinsten Teilfehlers liegt. Die Fehlerwerte selbst entsprechen Fehlerquadraten und sind daher grundsätzlich positiv. Das Addieren einer 1 sichert die Division ab.

$$\text{Fehler}_i = \frac{6}{\frac{ga}{fa_i+1} + \frac{gb}{fb_i+1} + \frac{gc}{fc_i+1}} \; ; \quad i = 1, \ldots, M \cdot P \tag{6.3}$$

$$fa_i = \text{i-ter Fehler der Eigenwertberechnungen} \tag{6.4}$$

$$fb_i = \text{i-ter Fehler der Hopfield-Netze} \tag{6.5}$$

$$fc_i = \text{i-ter Fehler der Matching-Verfahren} \tag{6.6}$$

$$ga = \text{Gewichtung der Eigenwertberechnung} \tag{6.7}$$

$$gb = \text{Gewichtung des Hopfield-Netzes} \tag{6.8}$$

$$gc = \text{Gewichtung des Matching-Verfahrens} \tag{6.9}$$

6.3 Aufteilung der Algorithmen auf zwei Prozessoren

Die Entwicklung der Software für das Erkennungssystem erfolgte für einen Zielrechner, der ein Dual-Prozessor-Board besitzt. Damit ist es möglich, die zeitkritischen Module auf dem einen Prozessor und die Module zur Verwaltung des Erkennungssystems auf dem anderen Prozessor ablaufen zu lassen. Die in Echtzeit zu erfolgende Gesichter-Erkennung wird daher von allen anderen Aufgaben befreit und kann somit in minimaler Zeit ein Ergebnis liefern. Auch im Hinblick auf die Nachführung der Kamera muss die Rechenzeit der Erkennungsalgorithmen so gering wie möglich sein. Die Abb. 6.5 zeigt das Foto einer typischen Dual-Prozessor-Hauptplatine.

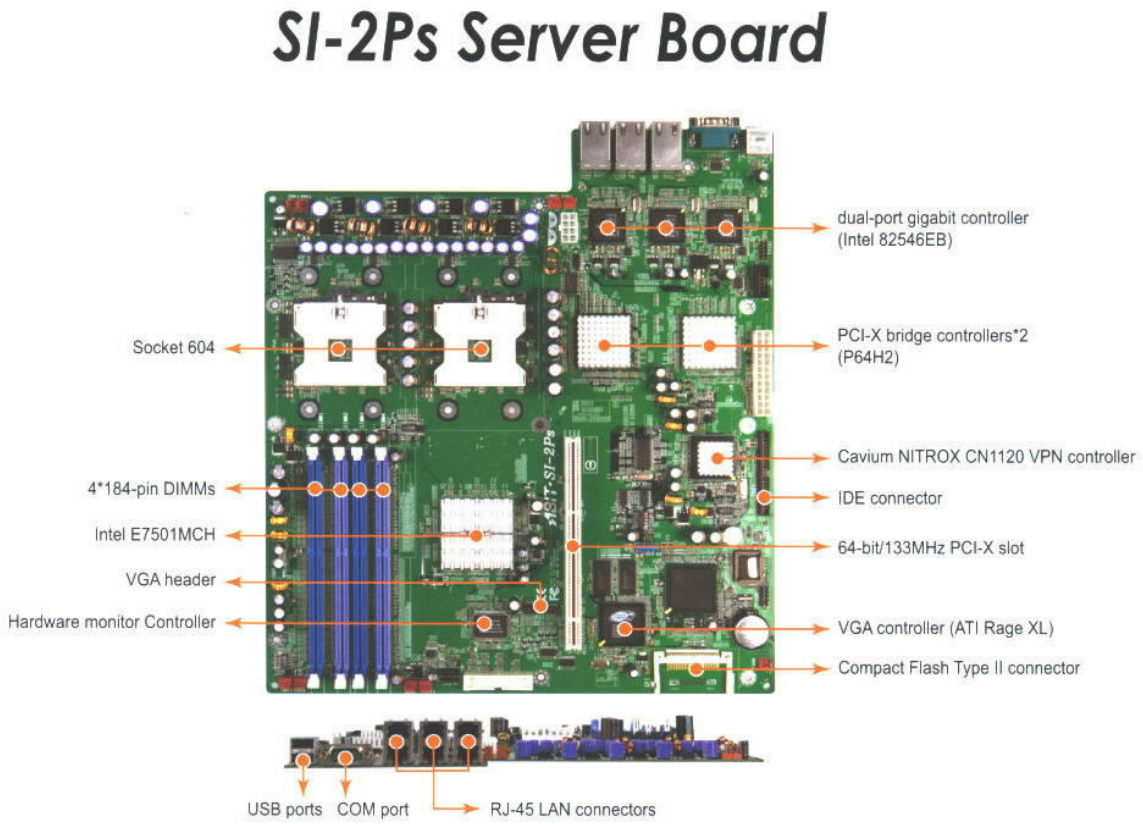


Abb.: 6.5: Ansicht eines Dual-Prozessor Board ohne Prozessoren

Die Programmierung zur Aufteilung der Rechenlast erfolgt nach dem „shared-memory“ Prinzip unter Verwendung eines speziellen frei verfügbaren Java-Programms, das wie ein Preprozessor arbeitet. Der Textfile des Java-Programms wird dazu nicht mit der standardisierten Endung „java“ gespeichert, sondern abweichend mit der Endung „jomp“. Das Preprozessor-Programm jomp.java.jomp liest diesen jomp-File ein, interpretiert alle Programmzeilen, die mit der Zeichenfolge

```
//omp
{
   ... Anweisungen
}
```

beginnen und erzeugt daraus automatisch einen neuen Code, der nun jedoch mit der vorgeschriebenen Endung „java“ abgelegt wird.

Der generierte Textfile mit der Endung „java“ läßt sich wie folgt starten:

```
java -Djomp.threads=2 filename
```

Das folgende Listung zeigt ein typische Programmsegment zur Bildung eines Mittelwertes mit dem Fall „reduction“. Die //omp parallel Anweisung definiert den parallelen Programmbereich. Das Zuordnen von Anfangswerten für die lokalen Variablen läßt den Preprozessor erfolgreich arbeiten.

```
static float Summation(int N, float Vector[])
 {
  int   i       =         0 ;
  float summe   =         0 ;
  float n       = (float) N ;
  float xmittel =         0 ;

  OMP.setThreadNum(2) ;

  //omp parallel private (summe) reduction(+:float xmittel)
   {
    //omp for
      for (int i = 0; i < N ; i++)
          {
           summe = summe + Vector[i] ;
          }
      xmittel = summe / (float) N ;
   }

  return xmittel ;
 }
```

Zum Kontrollieren der einzelnen n Threads dienen die folgenden Kommandos:

```
setNumThreads(int n)

n = getThreadNum()

getNumProcs()
```

Die Aufteilung der verschiedenen Methoden auf die einzelnen Prozessoren kann entweder mittels der sections bzw. section Anweisung

```
//omp parallel
  {
   //omp sections
     {
      //omp section {Methodenaufrufe für einen Prozessor        }
      //omp section {Methodenaufrufe für einen anderen Prozessor}
     }
  }
```

oder durch die direkte und somit definierte Zuordnung der Methoden auf die einzelnen Prozesoren erfolgen:

```
//omp parallel
  {
   if (getThreadNum == 0)
      {
       Methodenaufrufe für den Programmstart,
       den Update-Vorgang und
       die Bildschirmausgaben
      }
  if (getThreadNum == 1)
    {
     Methodenaufrufe für die echtzeitfähige Bilderkennung
    }
  }
```

Synchronisierungsprobleme treten beim realisierten Programm nicht auf, denn der Datenaustausch zwischen den Prozessoren findet nur zu definierten Zeitpunkten statt, z.B. bei einer Änderung der Anzahl der gespeicherten Bilder.

Die Abb. 6.6 ermöglicht einen Blick in den Zielrechner. Die beiden Prozessoren verstecken sich hinter den Lüftern. Den erhöhten Strombedarf liefert ein 600W-Netzteil, das links oben in der Abb. 6.6 zu erkennen ist.

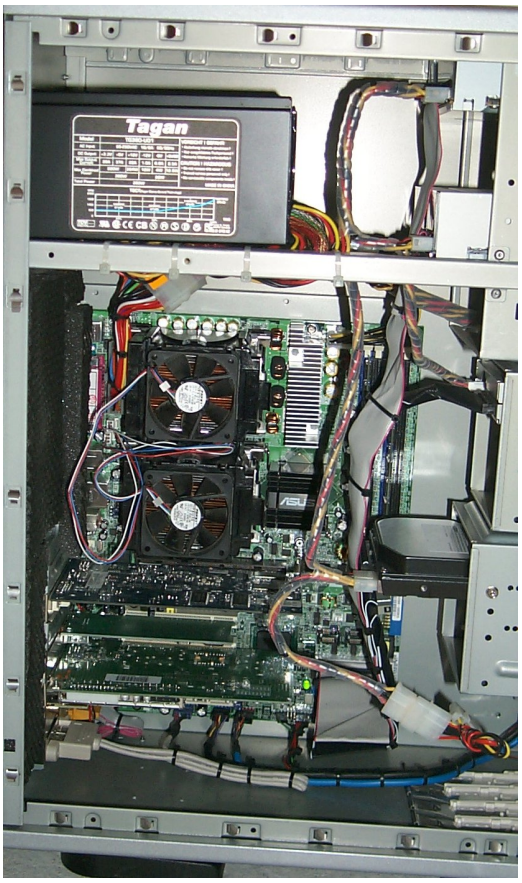


Abb.: 6.6: Ein Blick in einen der eingesetzten Rechner mit Dual-Processor Board

6.4 Ergebnisse für ein Referenzbild

Die Abb. 6.7 stellt 10 Gesichtsaufnahmen dar, die als Basis für das Erkennnungssystem dienen. Die Parallelisierung der Bilder erfolgt in zwei Zeilen von jeweils 5 Einzelbildern.

Abb.: 6.7: Test des Programms mit identischen Bildern

Das Matching-Verfahren erfordert die Generierung der signifikanten Bildpunkte, die zeilenweise in der linken Bildhälfte in den Übergängen von weiß nach schwarz und in der rechten Bildhälfte umgekehrt von schwarz nach weiß gesucht werden. Falls kein Übergang vorhanden ist, setzt der Algorithmus einen Punkt bei $\frac{1}{4}$ bzw. bei $\frac{3}{4}$ der Bildbreite. Die Abb. 6.8 zeigt eine Auswahl von Bildern und die entsprechend generierten roten Bildpunkte.

Abb.: 6.8: Generierung von Bildpunkten

Außerdem erfolgt beim Start des Programms die Berechnung der beiden Transformationsmatrizen für die Transformation in den Raum der Eigenbilder gemäß der Eigenwert-Methode. Die Eigenbilder speichert man zusammen mit den Transformationsmatrizen ab.

Die beiden Hopfield-Netze basieren auf der Berechnung der $(80 \times 60)^2 \cdot 2 = 46$ Millionen Gewichte, die ebenfalls zu speichern sind. Legt man die in der Abb. 6.7 dargestellten Gesichter zeilenweise an die beiden Hopfield-Netz an, dann tauchen nach wenigen Iterationen am Ausgang der neuronalen Netze die identischen Bilder als Ausgangssignal wieder auf.

Als Programmtest schaltet man nun ein Bild aus der Sammlung der Bilder zur Erkennung auf, d.h. das zu erkennende Bild stimmt mit einem der gespeicherten Bilder exakt überein. In diesem Fall erhält man die in der Abb. 6.9 dargestellten Einzelfehler. Die Buchstaben weisen auf das entsprechende Verfahren hin. Die Reihenfolge der Darstellung der Fehlerwerte entspricht den zunehmenden Gesamtfehlern.

Die Fehlerwerte der Eigenwert- und der Matching-Methode nehmen den Wert Null an. Das Hopfield-Netz erzeugt einen Rundungsfehler von Eins, der sich allerdings auf den Gesamtfehler nicht auswirkt. Das nächste Bild, die Person „nick“, weist schon einen Fehler von 18 auf.

Für das Bild der Person „nick“ stimmen die Werte der drei Einzelfehler zufällig überein. Daher ergibt die Kontrollrechnung gemäß der Gleichung (6.3) folgenden Gesamtfehler S:

$$\mathrm{S} = \frac{6}{\frac{1}{9} + \frac{1}{9} + \frac{1}{9}} = 18 \quad \longrightarrow \quad S18$$

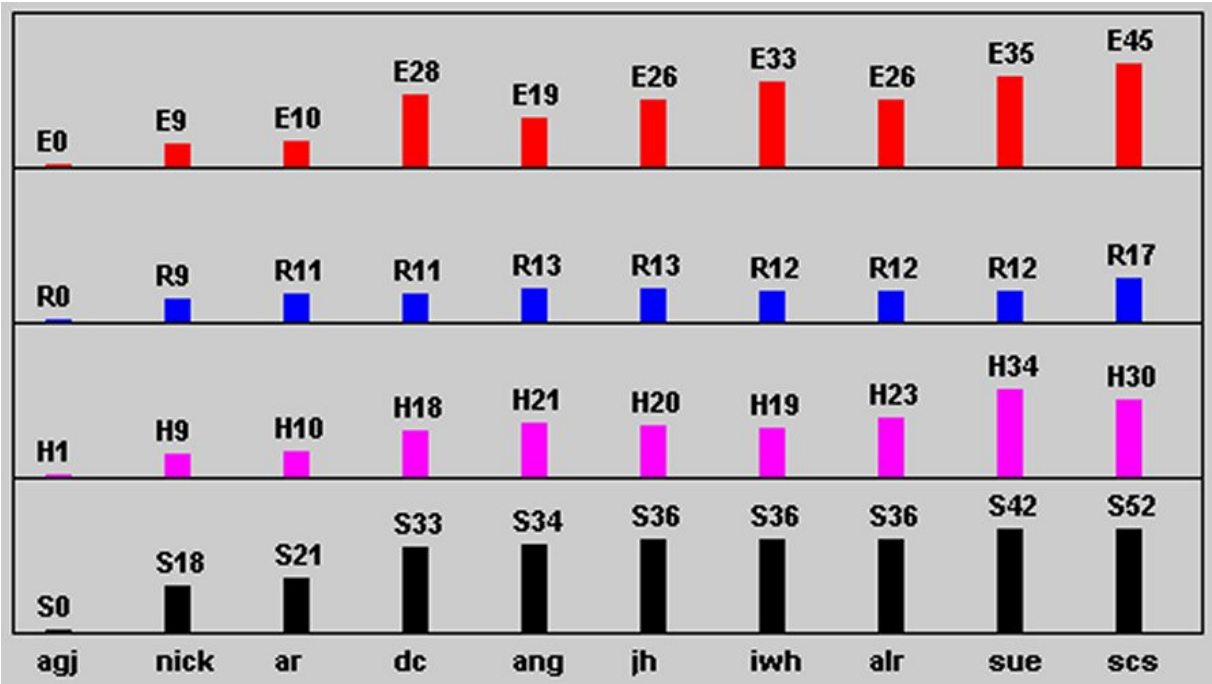


Abb.: 6.9: Fehlerbilanz eines bekannten Bildes

(E: Eigenwert-Methode, H: Hopfield-Netz,R: Matchingverfahren, S: Gesamtfehler)

Als hilfreiche Information über den Verlauf des Erkennungsverfahrens visualisiert die folgende Abb. 6.10 die einzelnen Zwischenschritte. Man erkennt, wie das Hopfield-Netz für das erkannte Bild das Eingangsbild direkt auf den Ausgang durchschaltet (unterste Bildreihe für 4 Iterationen des Hopfield-Netzes) und keine Veränderungen am Eingangsbild vornimmt. Dies ist ein wichtiger Test für die korrekte Arbeitsweise eines neuronalen Netzes, denn das Hopfield-Netz hat die angelegten Bilder „gelernt“ und kann sie daher in der Recallphase fehlerfrei reproduzieren. Dies ist bei der Vielzahl der Gewichtsfaktoren des neuronalen Netzes eine beachtliche numerische Leistung.

Die zweite Bildreihe stellt die Ergebnisse der Bildpunkt-Berechnung für die Matching-Methode dar. Die roten Bildpunkte gelten für die gespeicherten Referenzbilder und die blauen Bildpunkte ergeben sich nach der Rücktransformation des gescannten Bildes in die Ebene der Referenzbilder mit den jeweiligen geometrischen Transformationsmatrizen. Falls Bildpunkte übereinstimmen, wie dies für die erkannte Person „agj“ der Fall ist, lassen sich in der Darstellung nur noch blaue Bildpunkte erkennen.

Der Begriff „Hamming“ entspricht der mittleren Abweichung der einzelnen gespeicherten Bilder vom gescannten Bild und liefert Kontrollwerte für den Anwender. Für die gespeicherten Bilder sollte die Hamming-Differenz möglichst groß sein, damit der Bildraum optimal genutzt werden kann.

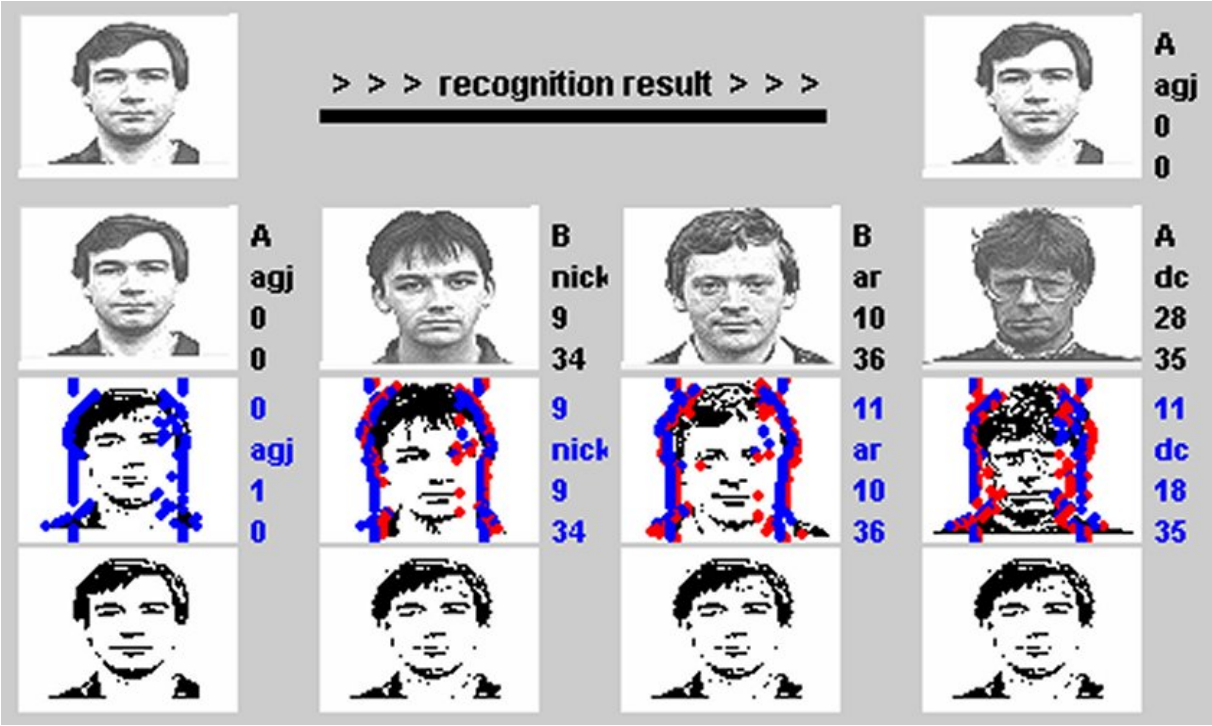


Abb.: 6.10: Ergebnisse des Erkennungsverfahrens für ein bekanntes Bild

Damit gilt nun das gesuchte Bild als erkannt und erscheint auf dem Bildschirm unterhalb der gescannten Aufnahme. Wie die Abb. 6.11 zeigt, liefert für diesen speziellen Testfall das Erkennungsverfahren selbstverständlich das gespeicherte Bild als erkanntes Bild zurück.

Abb.: 6.11: Gespeichertes Bild (oben) und erkanntes Bild (unten)

Der Test mit der Erkennung eines bekannten Bildes aus der Sammlung der gespeicherten Bilder diente zur Überprüfung der Algorithmen. In den folgenden Beispielen weichen die gescannten Bilder von den gespeicherten Bildern nun erheblich ab. Dazu werden verschiedene Fälle betrachtet.

Kapitel 7

Erprobung des Erkennungssystems

7.1 Erprobung mit Testbildern

1. Die Person „agj“ hält ein Glas vor den Mund

In der Abb. 7.1 hält die Person „agj“ ein Glas vor den Mund. Das Hopfield-Netz rechnet diese Störung des Bildinhaltes nach den ersten beiden Schritten heraus (siehe unterste Bildreihe). Die gesuchte Person führt bei allen drei Fehlerarten zum geringsten Fehlerwert.

Abb.: 7.1: Die Person „agj“ mit Glas

2. Die Person „agj“ hält die Hand zum Gruss an die Stirn

Wie an der Abb. 7.2 zu erkennen ist, wird die Person „agj“ auch dann sicher erkannt, wenn sie die Hand zum Gruss an die Stirn hält. Das gute Ergebnis von H8 des Hopfield-Netzes löst einen sehr geringen Wert für das Regressionsverfahren von R12 aus, der weit unter den anderen R-Werten liegt.

Deutlich erkennt man in der untersten Bildfolge, wie das Hopfield-Netz nach zwei Schritten die Hand aus dem Bild entfernt. Dabei kommt es dann zu einer Störung des Bildinhaltes, die zu einem Fehler H8 führt.

Die Eigenwert-Methode liefert für die Personen „agj“, „ar“ und „nick“ nahezu identische Werte und wäre daher als eigenständiges Verfahren nicht in der Lage, die Person „agj“ sicher zu erkennen.

Abb.: 7.2: Die Person „agj“ mit Hand

Die graphische Darstellung der Fehlersäulen ist auf den maximalen Wert von 40 begrenzt. Falls, wie in der Abb. 7.2, die Fehlerwerte diesen Wert übersteigen, läßt sich der Fehlerwert an der Beschriftung oberhalb der Säulen-Darstellungen ablesen.

In der Bildreihe oberhalb der Hopfield-Bilder markieren die roten Punkte das Referenzbild und die blauen Punkte die auf die Ebene der Referenzbilder zurück transformierten gescannten Bilder. Im günstigsten Fall stimmen beide Bildpunktfolgen überein und dann dominiert die blaue Farbe.

3. Die Person „agj“ vergrößert und mit Hintergrund

Die Abb. 7.3 stellt die vergrößert gescannte Person „agj“ mit einer weiteren Person im Hintergrund dar. Die Eigenwert-Methode fällt auf diesen Hintergrund herein und berechnet einen Fehlerwert von E26, der gegenüber den Werten E15 für die Personen „iwh“ und „dc“ deutlich erhöht ist. Das Hopfield-Netz beseitigt den Hintergrund und ermöglicht in Verbindung mit der Matching-Methode die erfolgreiche Erkennung der Person „agj“ auf der Basis der Fehlerwerte H11 und R12. Der resultierende Gesamtfehler S28 unterscheidet sich deutlich gegenüber dem nächst größeren Wert von S34.

Die Rechenzeit beträgt bei Start des Programmes ca. 24 s. Der eigentliche Erkennungsprozess benötigt nur noch 1,3 s. Der Versuch, die Erkennungszeit noch weiter durch Parallelisierung zu reduzieren, führt zu einer weiteren Erhöhung des ohnehin schon sehr hohen Speicherbedarfes. Bei der vorhandenen Hardware schlug dieser Versuch schnell fehl, denn das Betriebssystem lagerte dann verstärkt Speicherbereiche auf die Festplatte aus und das gefürchtete „swappen“ setzte ein. Der enorm hohe Speicherbedarf, der hauptsächlich von den Gewichten der beiden Hopfield-Netze und dem Regressionsverfahren benötigt wird, erfordert zusätzliche Parameter für die Bereitstellung des erweiterten Speicherbedarfs beim Start des Main-Programms:

```
java mainprogram -Xms1024m -Xmx1024m
```

Abb.: 7.3: Die Person „agj“ vergrößert und mit Hintergrund

4. Die Person „agj“ schräg aufgenommen

Die Abb. 7.4 stellt die vergrößert und schräg gescannte Person „agj“ dar. Das Hopfield-Netz kompensiert die seitliche Aufnahme und dreht den Kopf wieder zurück, was sich auf die Erkennung sehr günstig auswirkt. Mit dem geringen Fehlerwert von H11 für die Person „agj“ und H10 für die Person „nick“ bietet das Ausgangsbild des Hopfield-Netzes trotz der zunächst falschen Reihenfolge bei der Erkennung der Personen eine gute Voraussetzung für das Matching-Verfahren, das sich mit einem Fehler von nur noch R12 gegenüber R40 klar für die Person „agj“ entscheidet.

Abb.: 7.4: Die Person „agj“ von der Seite aufgenommen

Die dargestellten jeweils vier Bildfolgen, die sich oberhalb der Hopfield-Bildreihe befinden, entsprechen den Referenzbildern in der Reihenfolge der geringsten Abweichung zum gescannten Bild. Der überwiegende Anteil an blauen Punkten in der Darstellung des Matching-Verfahrens für die Person „agj“ lässt schon erkennen, dass in diesem Fall der Regressionsalgorithmus eine weitgehende Übereinstimmung mit dem gespeicherten Bild erzielen konnte.

An dem Matchingbild der Person „iwh“ fällt auf, dass es offenbar schwierig ist, in der dichten Haarkrause die Bildpunkte eindeutig zu setzen. Daher kommt der Vorfilterung der Bilder die Aufgabe zu, geschlossene schwarze Flächen aufzulösen und durch Randlinien zu ersetzen. Der eingesetzte Laplace-Filter kann diese Aufgabe hervorragend lösen.

5. Die Person „ang“ erscheint mit Brille

In der Abb. 7.5 erscheint die Person „ang“ gegenüber dem gespeicherten Bild mit einer dunklen Brille vor der Überwachungskamera. Wie zu erkennen ist, erfolgt die Erkennung der Person „ang“ ohne Probleme mit einem Fehler von S24 gegenüber dem nachfolgenden Fehlerwert von S28. Das Hopfield-Netz beseitigt die Brille nach drei Schritten und liefert einen Fehler von H13. Die Störung des Bildinhaltes, welche das Hopfield-Netz verursacht, führt allerdings dazu, dass die Person „agj“ mit einem geringeren Fehler von H12 am Ausgang des Hopfield-Netzes erscheint.

Die Matching-Methode, die für die Frau mit Brille einen Fehlerwert von R32 ergibt, liegt in der Erkennung erheblich schlechter als die Werte für die Person „agi“, „dc“, „iwh“ und “nick“.

Die erfolgreiche Erkennung der richtigen Person „ang“ basiert schließlich auf dem geringen E7 Fehlerwert der Eigenwert-Methode.

Abb.: 7.5: Die Person „ang“ mit Brille

Dieses Beispiel verdeutlicht das optimale Zusammenspiel der verschiedenen Erkennungsalgorithmen. Ein geringer Fehlerwert bedeutet immer eine erfolgreiche Erkennung für das jeweilige Verfahren und muss daher den Ausschlag für das Endergebnis stark beeinflussen. Falls ein bestimmtes Verfahren einen hohen Fehlerwert liefert, dann bedeutet das nicht zwangsläufig, dass die gescannte Person unbekannt ist. Dagegen gilt aber umgekehrt: Ein sehr geringer Fehler kann nur dann auftreten, wenn das gescannte Gesicht mit dem gespeicherten Gesicht übereinstimmt.

6. Die Person „ang“ hält den Kopf schräg

In der Abb. 7.6 hält die Person „ang“ den Kopf schräg. Das Hopfield-Netz rechnet die schräge Kopfstellung zurück und erkennt sicher die Person „ang“. Ähnlich wie bei der Person mit Brille sichert die Eigenwert-Methode wieder den Erfolg der Erkennung.

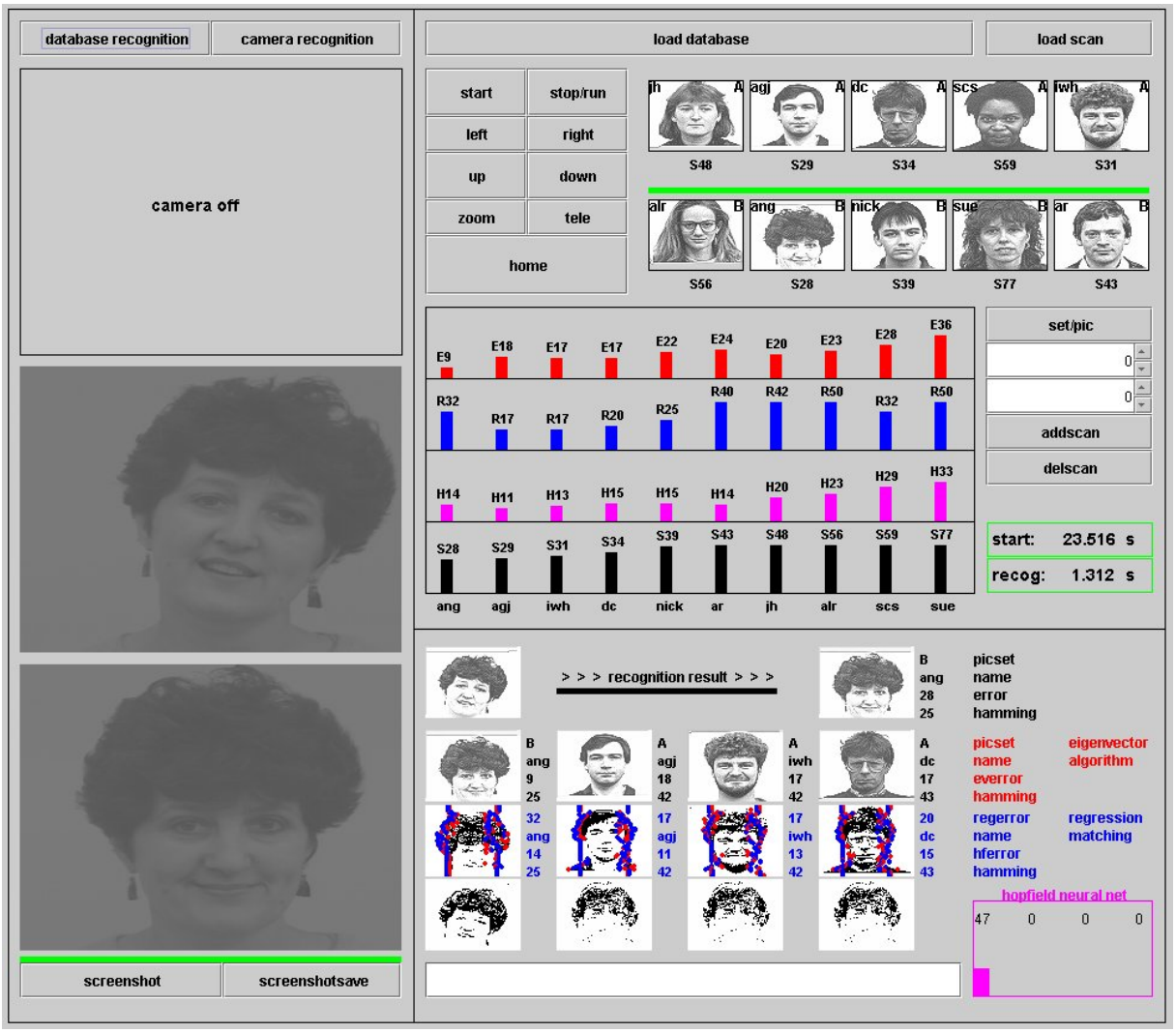


Abb.: 7.6: Die Person „ang“ mit schräger Kopfhaltung

Interessant an diesem Fall einer Gesichtserkennung ist die Beobachtung der Ausgangsbilder des Hopfield-Netzes. Bereits nach dem ersten Schritt richtet sich der Kopf wieder auf. Leider muss auch in diesem Fall eine Verfälschung des Bildinhaltes hingenommen werden. Erhalten bleibt jedoch die äußere Kontour dieses Gesichtes, so dass die Matching-Methode eigentlich ein gutes Ergebnis liefern müsste. Dies ist aber nicht der Fall, denn der Fehler erreicht den großen Wert von R32.

Die Ursache dieser Beobachtung liegt in der Schwierigkeit, den Bereich der Frisur mit einer signifikanten Folge von Bildpunkten zu beschreiben. Innere Teilbereiche der Frisur weisen einen höheren Kontrast als die eigentlichen Randlinien auf und schwächen dadurch die Leistungsfähigkeit der Matching-Methode stark ab.

Eine Anpassung der Filterparameter würde in diesem Beispiel - allerdings auf Kosten anderer Beispiele - zu besseren Ergebnissen führen.

7. Die Person „scs“ mit Tasse

Ein Blick auf die unterste Bildfolge der Ausgangsbilder des Hopfield-Netzes in der Abb. 7.7 zeigt, dass bereits nach einem Iterationsschritt die Hand und die Tasse nahezu verschwunden sind. Das Hopfield-Netz eignet sich jedoch allein nicht zur Erkennung dieser Person „scs“, denn die Beseitigung der Tasse hat das Ausgangsbild stark verändert. Der entsprechende Fehlerwert H19 liegt deutlich über dem Fehlerwert von H8, der sich für die Person „nick“ ergibt. Die Matching-Methode rettet mit dem sehr geringen Fehlerwert von R12 zusammen mit dem guten Ergebnis für die Eigenwert-Methode von E11 die Situation.

Abb.: 7.7: Die Person „scs“ mit Tasse

Die Matching-Methode kann in diesem Beispiel ein gutes Ergebnis liefern, weil die Person „scs“ eine markante und geometrisch klar strukturierte Kopfform besitzt. Die Bildpunkte liegen exakt auf den äußeren Randlinien dieses Gesichtes, wie das erste Bild in der zweiten Bildreihe von unten erkennen läßt.

Auch dieses Beispiel verdeutlicht wieder die optimale Zusammenarbeit der unterschiedlichen Erkennungsalgorithmen. Bestimmte Formen und Strukturen von Gesichtsaufnahmen lassen sich mit bestimmten Erkennungsverfahren optimal auswerten. Für die Gewichtung der unterschiedlichen Fehleranteile zur Berechnung des für die Erkennung letztlich entscheidenden Gesamtfehlers geben wieder die Verfahren mit den kleineren Fehleranteilen den Ausschlag. Fehlgeschlagene Erkennungsversuche, die sich immer durch einen hohen Fehlerwert auszeichnen, wirken sich daher kaum auf den Gesamtfehler aus.

8. Die Person „nick“ lesend und hinter Gittern

Einen interessanten Fall zeigt die Abb. 7.8. Hier hält sich die Person „nick“ hinter einem Gitter auf und liest in einem Ordner. Das Erkennungssystem löst auch diese Aufgabe und liefert die richtige Person als erkannt zurück.

Gegenüber den beiden letzten Fällen hat nun die Eigenwert-Methode mit einem Fehler von E26 nur noch eine geringe Bedeutung im Erkennungsprozess. Vielmehr liefern die Hopfield-Methode mit H16 und die Matching-Methode mit R14 die geringsten Fehlerwerte.

Erstaunlich wie das Hopfield-Netz zunächst die Gitter beseitigt und dann den Ordner, in dem die Person „nick“ liest, abschwächt. Nach zwei Schritten ist das Hopfield-Netz bereits vollständig eingeschwungen.

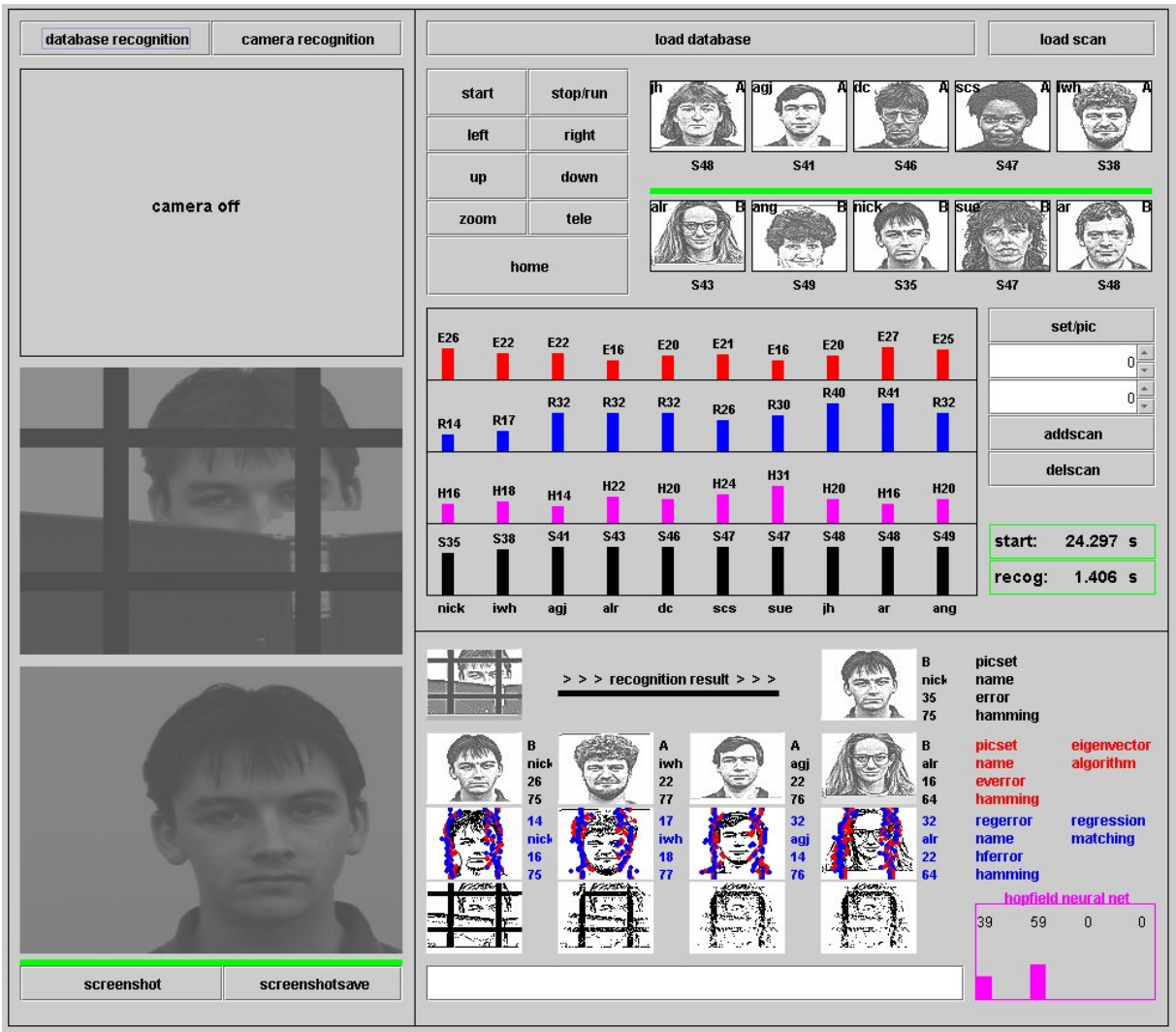


Abb.: 7.8: Die Person „nick“ lesend und hinter Gittern

Dieses Beispiel erreicht einen Schwierigkeitsgrad, bei der ein menschlicher Betrachter durch konzentriertes Vergleichen der Gesichtsaufnahmen gerade noch in der Lage ist, die richtige Person „nick“ zu erkennen.

Dem programmierten Erkennungsverfahren genügen auch signifikante Teilbereiche eines gescannten Gesichts, um das Gesicht eindeutig einer gespeicherten Gesichtsaufnahme zuordnen zu können.

9. Die Person „ar“ mit geschlossener Kapuze

Ein Blick auf die unterste Bildfolge der Ausgangsbilder des Hopfield-Netzes in der Abb. 7.9 zeigt, dass nach drei Iterationsschritten das Bild keine Kapuze mehr enthält. Im Ausgangsbild des Hopfield-Netzes ist nun die vermummte Person bereits strukturell erkennbar.

Die relativ geringen Fehlerwerte der Matching-Methode und der Eigenwert-Methode liefern einen Gesamtfehler, der mit S20 gerade noch unterhalb des nächst größeren Fehlers für die Person „agj“ liegt und somit eine richtige Erkennung ermöglicht.

Abb.: 7.9: Die vermummte Person „ar“ mit geschlossener Kapuze

Dieses Ergebnis ist besonders eindrucksvoll. Zeigt es doch, welch enorme Leistungsfähigkeit die Kombination der verschiedenartigen Erkennungsalgorithmen besitzt.

Dem menschlichen Betrachter wird es schwer fallen, in diesem Beispiel die richtige Person zu erkennen. Das Hopfield-Netz dagegen hat ohne große Mühen die Kapuze entfernt und das unvermummte Gesicht als Ausgangssignal zur weiteren Bearbeitung geliefert.

Das hervorragende Ergebnis, das sich in diesem Beispiel zeigt, demonstriert in besonders deutlicher Weise das hohe Potential, das die optimale Kombination verschiedener Erkennungsverfahren besitzt.

7.2 Erfassen von Kamerabildern

Die Abb. 7.10 zeigt die Bedienelemente zur Ansteuerung der Kamera. Damit lassen sich neue Aufnahmen erstellen. Zunächst richtet man die Kamera auf das zu erfassende Gesicht aus und stellt mit den Knöpfen „Zoom“ bzw. „Tele“ den optimalen Abstand ein. Das Gesicht sollte den dargestellten Bildausschnitt vollständig ausfüllen. Mit den Bedienelementen „screenshot“ friert man das momentane Kamerabild ein. Vor der Abspeicherung des fixierten Bildes wählt man noch einen Filenamen, der ohne Extension im untersten Textfeld einzutippen ist. Schließlich legt man noch die Bildzeile („set“) und die Bildnummer („pic“) fest. Nachdem diese Festlegungen erfolgt sind, startet man mit „screenshotsave“ das Abspeichern des Bildes. Das Erfassen und das Einsortieren eines ScreenShots erfordert somit die folgenden Schritten:

1. Ausrichten der Kamera
2. „load database“ betätigen
3. „screenshot“ betätigen
4. Filenamen ohne Extension in das unterste Textfeld eintragen
5. Bildzeile mit „set“ wählen
6. Bildnummer mit „pic“ wählen
7. Bedienelement „screenshotsave zum Abschluss betätigen

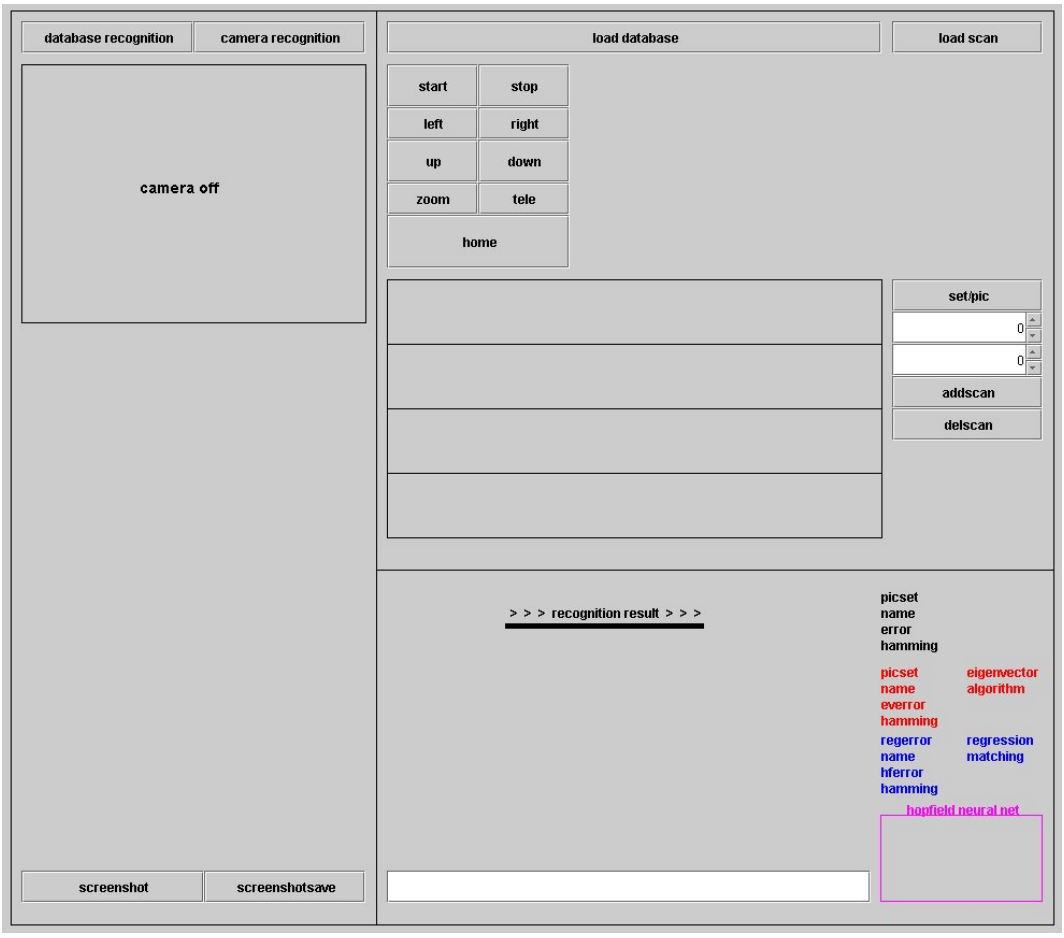


Abb.: 7.10: Steuerung der Kamera

Nach dem Abspeichervorgang starten die update-Algorithmen für die mit „set“ gewählte Bildzeile. Nach einigen Sekunden Rechenzeit ist das neue Bild in das Erkennungssystem integriert. Das neue Bild verdrängt das alte Bild von der Position „set“ bzw. „pic“. Das alte Bild bleibt auf der Festplatte erhalten, denn es ändert sich nur der Eintrag in der Liste der Filenamen für die einzelnen Bilder.

Der nachfolgende Textfile zeigt ein Beispiel für eine Liste mit dem Filenamen „AFileList.txt“ mit einem neuen Bild „wer.jpg“ mit der von Null aus zählenden Nummer pic = 2. Der eingegebene Filename „wer“ erscheint in der Fileliste mit der Erweiterung „-10.jpg“. Die Klasse „Parameter“ enthält die entsprechenden Pfadangaben.

```
E:\Bilderkennung\PickUpSystem\DatabaseA\jh-10.jpg
E:\Bilderkennung\PickUpSystem\DatabaseA\agj-10.jpg
E:\Bilderkennung\PickUpSystem\DatabaseA\wer-10.jpg
E:\Bilderkennung\PickUpSystem\DatabaseA\scs-10.jpg
E:\Bilderkennung\PickUpSystem\DatabaseA\iwh-10.jpg
E:\Bilderkennung\PickUpSystem\DatabaseA\pps-10.jpg
E:\Bilderkennung\PickUpSystem\DatabaseA\zxp-10.jpg
E:\Bilderkennung\PickUpSystem\DatabaseA\phil-10.jpg
```

Der ASCII-File „BFileList.txt“ enthält die Filenamen für die zweite Reihe der Bilder:

```
E:\Bilderkennung\PickUpSystem\DatabaseA\alr-10.jpg
E:\Bilderkennung\PickUpSystem\DatabaseA\ang-10.jpg
E:\Bilderkennung\PickUpSystem\DatabaseA\nick-10.jpg
E:\Bilderkennung\PickUpSystem\DatabaseA\sue-10.jpg
E:\Bilderkennung\PickUpSystem\DatabaseA\ar-10.jpg
E:\Bilderkennung\PickUpSystem\DatabaseA\jim-10.jpg
E:\Bilderkennung\PickUpSystem\DatabaseA\mjb-10.jpg
E:\Bilderkennung\PickUpSystem\DatabaseA\pds-10.jpg
```

Die Abb. 7.11 stellt die in den Filelisten gesammelten Bilder graphisch dar.

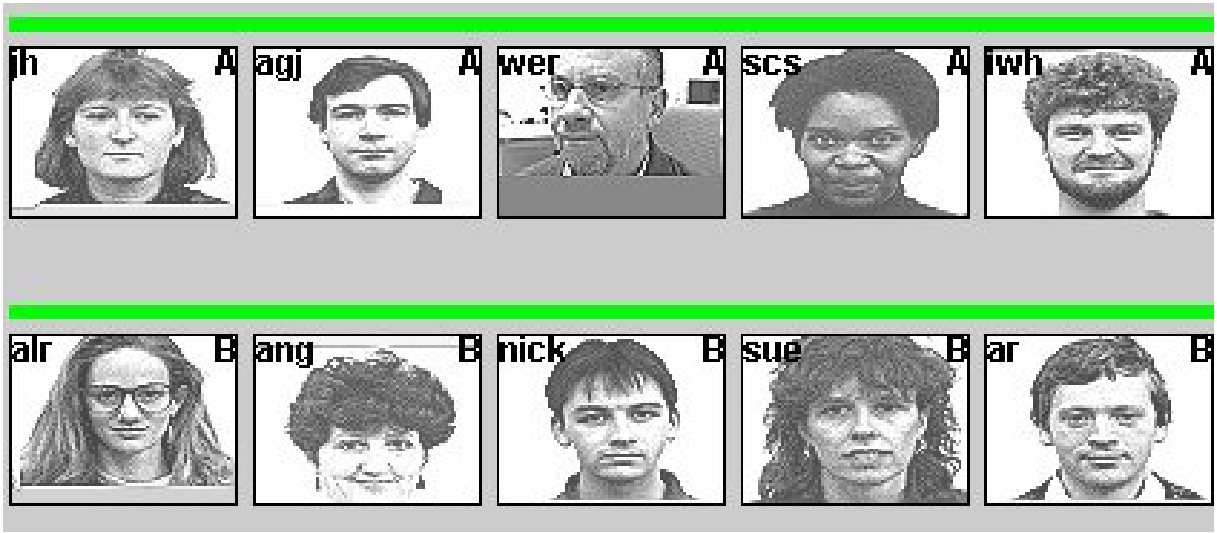


Abb.: 7.11: Darstellung der Bilder der Listen „AFileList.txt“ und „BFileList.txt“

7.3 Erkennung vermummter Personen

Das Bedienelement „camera recognition“ startet den dynamischen Test des Erkennungssystems. In einer Rechenzeit von ca. einer Sekunde vergleicht das Erkennungssystem auf der Basis der Eigenwert-Methode, des Hopfield-Netzes und des Regressionsverfahrens das Gesicht vor der Kamera mit allen gespeicherten Bildern.

Falls der ermittelte minimale Gesamtfehler einen in der Klasse „Parameter“ festgelegten Grenzwert einhält, gilt das Kamerabild als erkannt und die Referenz-Darstellung des entsprechenden Bildes mit dem geringsten Gesamtfehler erscheint im unteren linken Fenster.

1: Erkennung einer Person

Die Abb. 7.12 stellt ein Gesicht, das mit der Kamera erfasst wurde, dar. Die Erkennung der richtigen Person gelingt problemlos. Wie an der untersten Bildreihe der Abb. 7.12 zu erkennen ist, entfernt das neuronale Netz einen Teil der Kappe und erkennt in der gescannten Aufnahme nicht die richtige Person „w2“, sondern liefert für die Person „nick“ den geringeren Fehlerwert. In der Folge hätte sich das Regressionsverfahren für die Person „nick“ entschieden. Die tatsächlich erzielte richtige Lösung basiert auf dem geringen Fehler der Eigenwert-Methode, der sich in der Berechnung des Gesamtfehlers stark auswirkt.

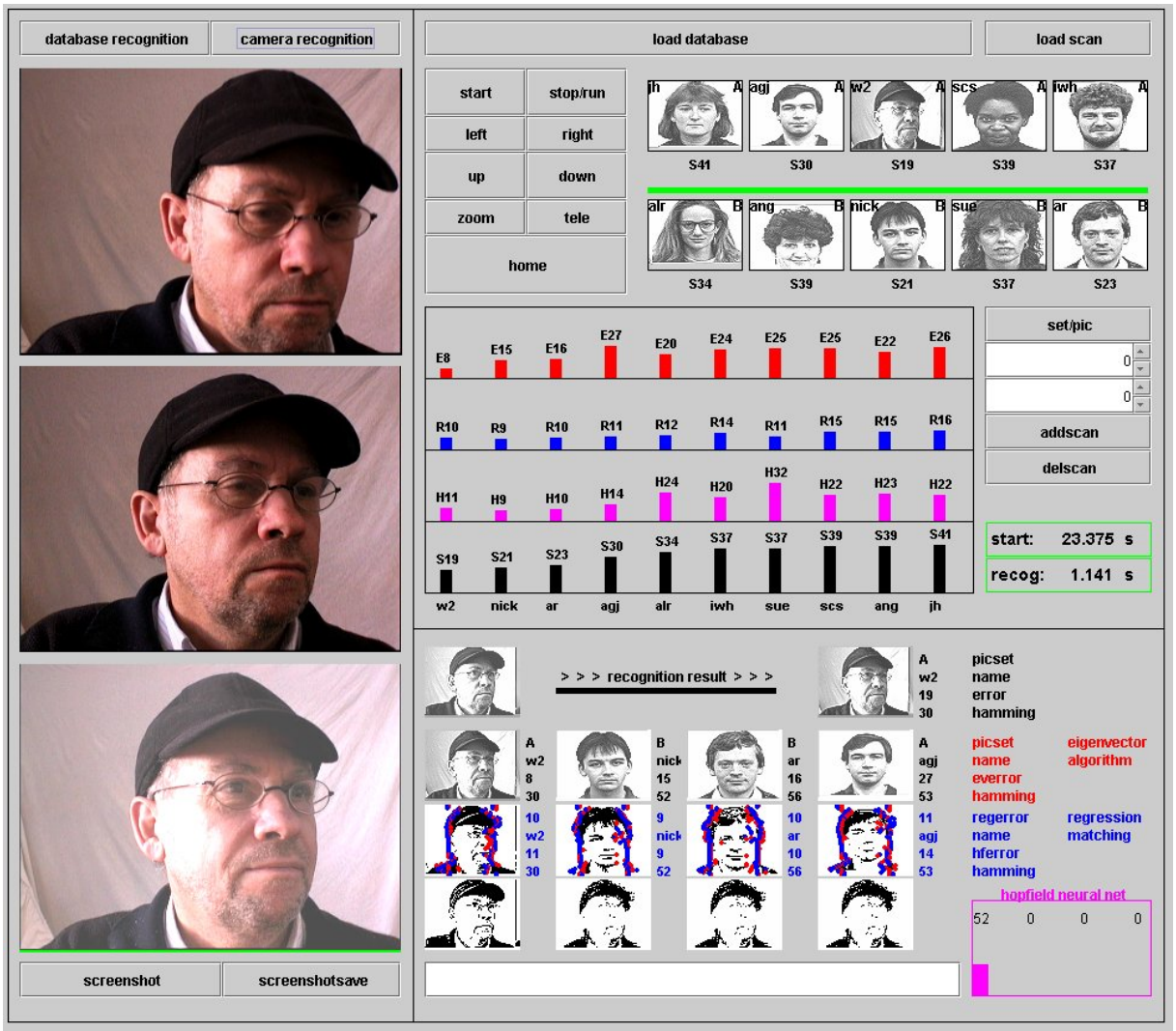


Abb.: 7.12: Gesicht einer nicht vermummten Person wird erkannt

Das gescannte Gesicht in der Abb. 7.12 weist zwar auf den ersten Blick eine weitgehende Übereinstimmung mit dem entsprechenden gespeicherten Bild auf, doch für das neuronale Netz unterscheiden sich die beiden Gesichtsaufnahmen erheblich.

Legt man zum Test an das neuronale Netz tatsächlich das gespeicherte Gesicht an, dann zeigt das Hopfield-Netz das erwartete und in der Abb. 7.13 dargestellte Verhalten.

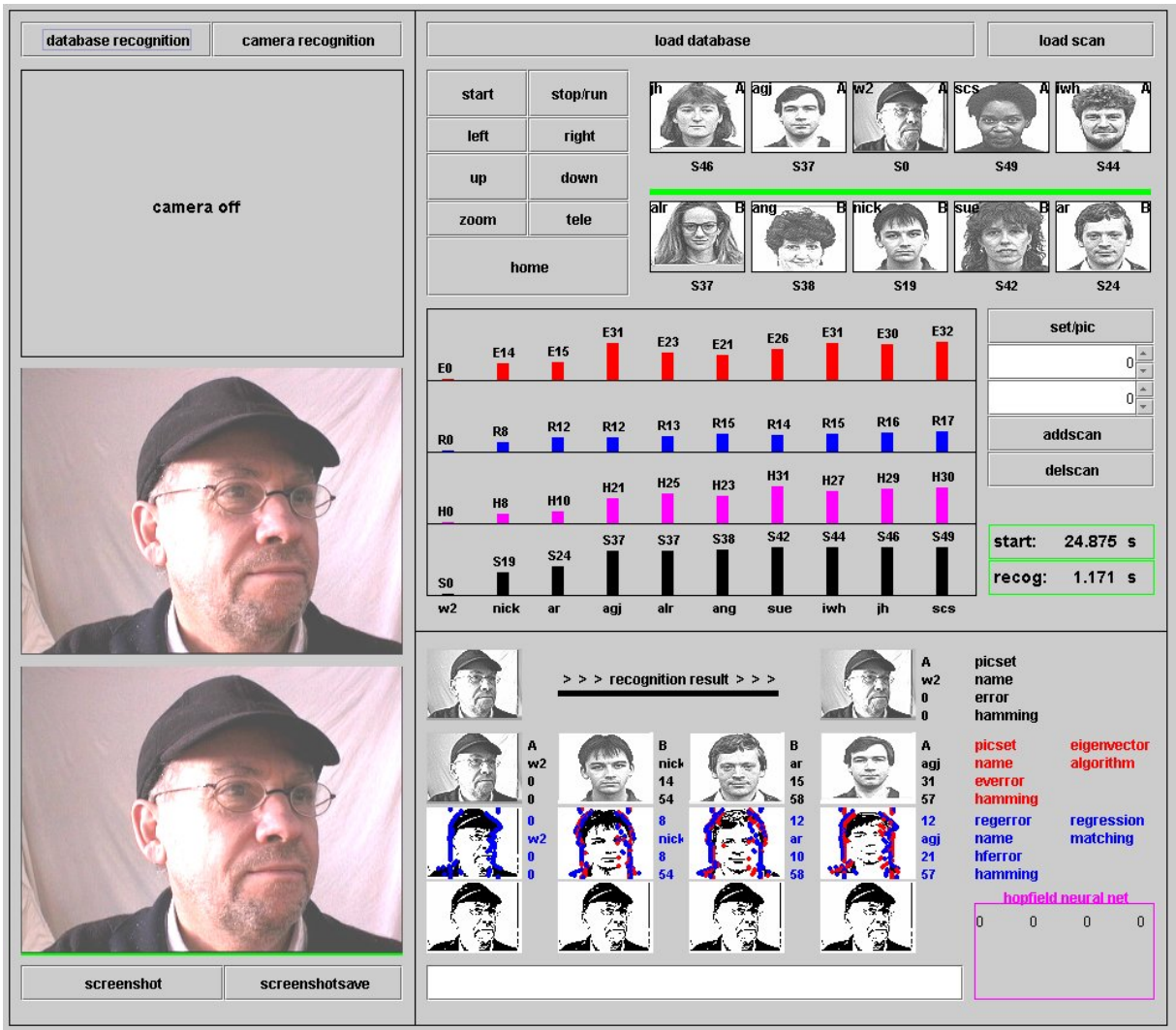


Abb.: 7.13: Erkennung einer identischen gespeicherten Person

Alle Erkennungsalgorithmen ergeben für diesen Sonderfall den Fehlerwert 0. Der Vergleich der Bildpunkte in der ersten Abbildung in der zweiten Bildreihe von unten läßt nur noch blaue Punkte erkennen, d.h. die roten Punte des gespeicherten Bildes liegen ausnahmslos exakt unterhalb der Bildpunkte, die aus der gescannten Aufnahme generiert und mit der Regressionsmethode zurück transformiert wurden.

Die Rechenzeit zur Bilderkennung reduziert sich bei der direkten Auswertung der Aufnahmen der Kamera von 1,3 s auf 1,1 s, denn dieser Vorgang erfordert ca. 200ms weniger Rechenzeit als das Einlesen der Bilder aus einer Bilddatenbank mit gescannten Aufnahmen.

Dieses Beispiel bestätigt die selbstverständliche Erkenntnis, dass jedes noch so triviale Erkennungssystem hervorragende Ergebnisse liefert, wenn der Unterschied zwischen gespeicherten und den gescannten Aufnahmen nur klein genug ausfällt.

Beispiel 2: Erkennung einer gespeicherten Person mit Sonnenbrille

Die Abb. 7.14 stellt die gescannte Aufnahme eines Gesichtes dar, das sich durch eine dunkle Sonnenbrille von der entsprechenden gespeicherten Aufnahme unterscheidet. Auch in diesem Beispiel demonstriert das Hopfield-Netz seine speziellen Eigenschaften, nämlich einerseits die störende Sonnenbrille zu eliminieren und andererseits wiederum das Bild zu verfälschen. Wieder rettet die Eigenwert-Methode die Situation und die richtige Person wird erkannt.

Abb.: 7.14: Gesicht mit einer Sonnenbrille wird erkannt

Als Problem in diesem Beispiel erkennt man die Schwierigkeiten beim Generieren der Bildpunkte für die äußere rechte Gesichtshälfte. Der Rand der Kappe, die Brille und der dunkle Punkt des rechten Auges führen zu erheblichen Abweichungen beim Matchingverfahren.

Die Eigenwert-Methode dagegen hängt von den generierten Bildpunkten nicht ab, sondern verwendet die Struktur des Bildes insgesamt als Basis für die Berechnung der entscheidenden L-Matrix, die wiederum innere Abhängigkeiten zwischen dem gescannten Bild und den gespeicherten Bildern in Form von Korrelationsparametern beschreibt. Gesichter, die für den menschlichen Betrachter in etwa gleichartig aussehen, führen daher bei der Eigenwert-Methode zu einem geringen Fehlerwert.

3: Erkennung eines vermummten Gesichtes

Die Abb. 7.15 stellt die gescannte Aufnahme eines Gesichtes dar, das mit einem Tuch stark vermummt ist. Die richtige Lösung basiert wieder auf der Eigenwert-Methode, deren geringer Fehleranteil gegenüber den anderen Methoden im Gesamtfehler überwiegt.

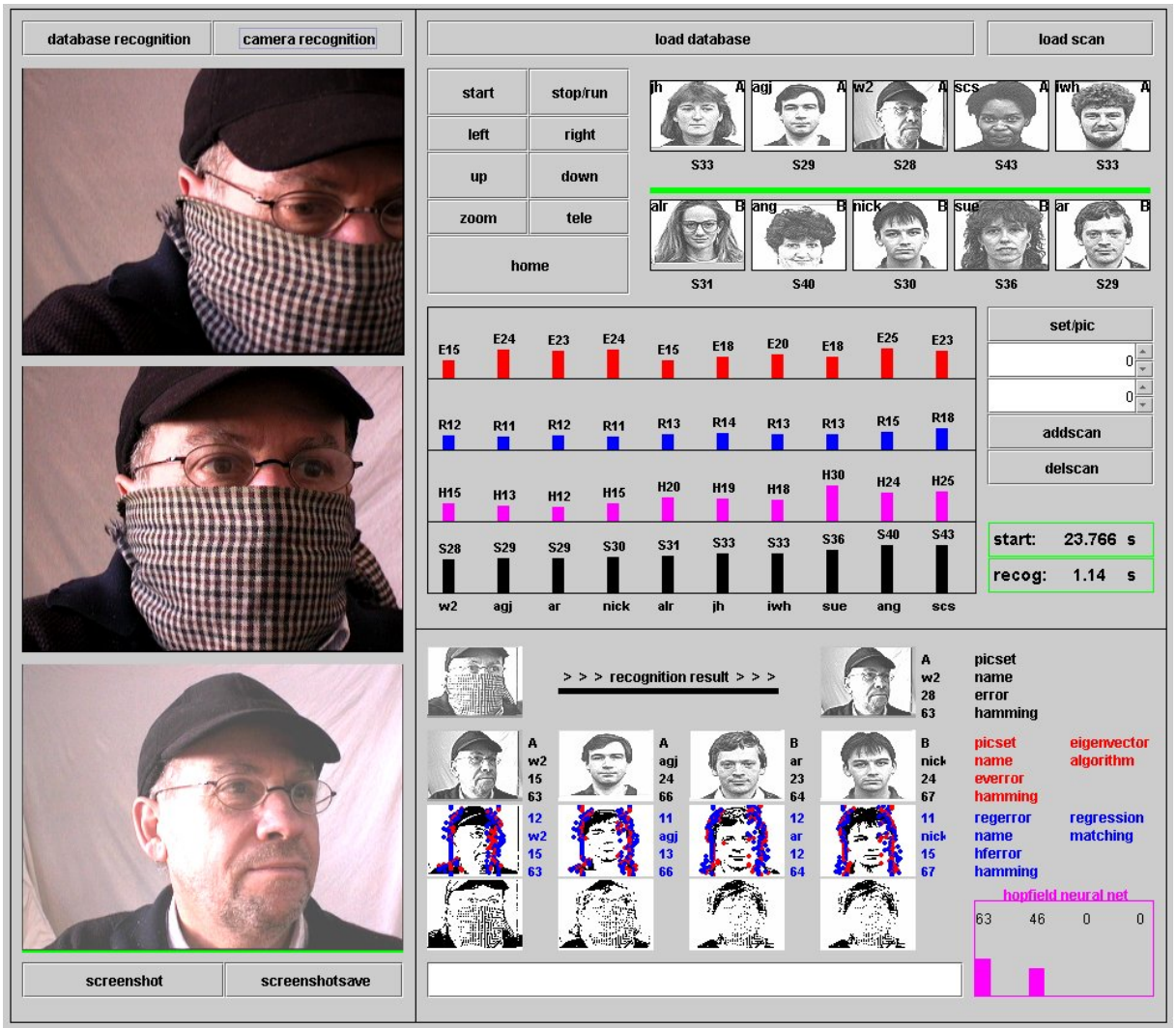


Abb.: 7.15: Vermummtes Gesicht wird erkannt

Interessant an dieser Gesichtsaufnahme ist das Verhalten des Hopfield-Netzes, das in nur zwei Schritten das karierte Tuch aus dem gescannten Gesicht fast ganz entfernt. Das dadurch entstehende Bild weist jedoch die größte Überstimmung nicht mit der richtigen Person „w2“ auf, sondern passt mit einem Fehler von H13 besser zur Person „agj“. Auch die gedrehte Kopfstellung der Person „w2“ fehlt im Ausgangsbild des Hopfield-Netzes. Man könnte also sagen: Das Hopfield-Netz hat zwar das karierte Tuch beseitigt, das Bild dabei aber so verfälscht, dass die falsche Person erkannt wurde. Der Unterschied zu den anderen Gesichtern „agj“, „ar“ und „nick“ ist jedoch so gering, dass auch die nachfolgende Matching-Methode keine sichere Unterscheidung mehr leisten kann und mit R11 und R12 fast identische Fehlerwerte liefert. Die Eigenwert-Methode erkennt jedoch die weitgehend vergleichbare Struktur des gescannten Bildes mit dem gespeicherten Bild der Person „w2“ und gibt daher mit dem Fehlerwert E15, der sich von den am nächsten liegenden Fehlerwerten E23 und E24 deutlich abhebt, den Ausschlag bei der Erkennung der richtigen Person „w2“.

7.4 Tracking-Betriebsart des Erkennungssystems

Schaltet man vor der Aktivierung des Bedienknopfes „camera recognition“ die Betriebsart „stop/run“ ein, dann startet eine Endlosschleife zur automatischen Nachführung der Kamera und zum Erkennen der Kamerascans in einer Endlosschleife. Ein Filteralgorithmus ermittelt im Takt von ca. einer Sekunde laufend die Schwerpunktkoordinaten des momentanen gescannten Bildinhaltes und interpretiert die Differenz zum geometrischen Bildmittelpunkt als eine zu minimierende Regelabweichung für die Kamerasteuerung. Dadurch richtet sich die Orientierung der Kamera auf das graphische Objekt aus. Die Abb. 7.16 zeigt eine Reihe von Bildschirmausschnitten für den Versuch einer Person, dem Erfassungsbereich der Kamera zu entkommen.

Abb.: 7.16: Beispiel für das Tracking einer Person

In einem weiteren Test sitzt dagegen die Person ganz ruhig und versucht sich so wenig wie möglich zu bewegen. Wie in der Abb. 7.17 zu sehen ist, befindet sich die Person beim Einschalten der Kamera am oberen rechten Bildrand.

Abb.: 7.17: Gesicht wird verfolgt und dann erkannt

In der unteren Bildreihe erkennt man, wie die Kamera einen Abwärts- und einen Seitenschwenk ausführt und dadurch das graphische Objekt automatisch in die Bildmitte rückt.

Während des eigentlichen Trackingvorganges darf noch keine Erkennung erfolgen, denn die schlecht zentrierten Aufnahmen würden zu hohen Fehlerwerten bei der Eigenwert-

Methode führen. Erst wenn die Zentrierung des graphischen Objektes eingeschwungen ist, starten die Erkennungsalgorithmen. Das Ende des Einschwingvorganges läßt sich an vernachlässigbaren Abweichungen des Mittelwertes des graphischen Objektes von der geometrischen Bildmitte erkennen.

Bewegt sich die Person, dann geht die Zentrierung zunächst wieder verloren und die Kamera wird erneut nachgeführt. Während der Dauer der Kamera-Nachführung pausieren die Erkennungsalgorithmen dann wieder .

Die Abb. 7.17, die Bildschirmausschnitte darstellt, zeigt in den oberen beiden grauen Bildreihen die Sammlung der gespeicherten Aufnahmen. Rechts oben in der Abb. 7.17 ist das erkannte Bild in der gespeicherten Form zur Kontrolle dargestellt.

In der linken Hälfte der Abb. 7.18 sieht man, wie die Kamera beginnt, das graphische Objekt zu zentrieren. Die Gesichtserkennung ist daher unterbrochen. Zur Kontrolle gibt diese Programmversion jedoch den berechneten Gesamtfehler an Stelle des erkannten Gesichtes aus. In der weißen Fläche lässt sich ein Gesamtfehler von 25 ablesen. Der maximale Fehlerwert, bei dem ein Gesicht überhaupt noch als erkannt gilt, beträgt 40.

Nachdem das graphische Objekt zentriert ist, erfolgt die Erkennung und die Darstellungen in der rechten Hälfte der Abb. 7.18. Das oberste Bild entspricht dem aktuellen Kamerabild, das mittlere Bild diente als Datenbasis für die Erkennungsalgorithmen und das Gesicht der erkannten Person ist in der untersten Darstellung zu sehen.

Abb.: 7.18: Pause der Erkennungsalgorithmen

Kapitel 8

Zusammenfassung und Ausblick

Auf Grund der zunehmenden Bedrohung der öffentlichen Sicherheit hat die Nachfrage nach einer automatischen Personenerkennung mittels Überwachungskameras stark zugenommen. Entsprechende Systeme sind weltweit im Einsatz. Der besondere Vorteil der Gesichtserkennung gegenüber allen anderen Kontrollverfahren besteht darin, dass die kontrollierten Personen selbst keine Aktionen ausführen müssen und sich unbemerkt im Vorbeigehen überprüfen lassen.

Die computergestützte Erkennung von Personen basiert auf der quantitativen Feststellung einer möglichst weitgehenden Übereinstimmung zwischen einer gespeicherten Graphik und dem momentan erfassten Kamerabild. Im Fall einer einfachen Anwendung gilt eine Person als erkannt, wenn eine gespeicherte Graphik dieser Person existiert, deren Quadratsumme aus den Differenzen der Pixelwerte ein absolutes Minimum annimmt und deren Abstand zum nächst größeren Fehlervektor ausreichend groß ist. Der direkte Bildvergleich setzt jedoch voraus, dass die zu vergleichenden Graphiken identische Aufnahmewinkel, Skalierungen, Beleuchtungsbedingungen, usw. aufweisen und vor dem gleichen Hintergrund aufgenommen werden. Doch als eigentliche Schwierigkeit kommt hinzu, dass die momentanen Kamera-Scans ein und derselben Person sehr unterschiedlich ausfallen können, denn die entsprechende Person kann ihr äußeres Erscheinungsbild im Vergleich zur gespeicherten Kameraaufnahme beliebig und vorsätzlich verändern.

Der mathematische Aufwand für die Erkennungsalgorithmen ist begrenzt, denn die eigentliche Erkennung soll in Echtzeit erfolgen. Weitere Bedingungen sind die Forderungen nach einer möglichst hohen Zuverlässigkeitsrate und die nachzuweisende Robustheit gegenüber jeder Art von Versuchen, das Erkennungssystem durch Täuschung zu überwinden.

Der vorliegende Forschungsbericht beschreibt ein Erkennungssystem, das die genannten Bedingungen durch die optimale Kombination unterschiedlicher, sich gegenseitig ergänzender Erkennungsverfahren erfüllt. Im Zentrum des Erkennungssystems arbeitet als wesentliche Komponente ein neuronales Netz, dessen Gewichte sich geschlossen berechnen lassen. Das Ausgangsbild des neuronalen Netzes gelangt anschließend als Eingangsbild zu einem Matching-Verfahren, das zum Vergleich der Bildpunkte die Hausdorff-Formel auswertet. Damit läßt sich das „non-rigid point matching“ vermeiden, die Ermittlung der Korrespondenz-Matrix entfällt.

Das auf Regression basierende Matching-Verfahren und das neuronale Netz bilden zwei der drei Fundamente, auf denen das Erkennungsverfahren aufgebaut ist. Das dritte und unverzichtbare Fundament liefert die Eigenwert-Methode, bei der mittels einer Transformationsmatrix alle Bilder in den „Eigenraum“ transformiert und dort auf Übereinstimmung überprüft werden. Die hervorragende Eigenschaft der Eigenwert-Methode besteht darin, dass sich ein prozentualer Wert für das Mass der Übereinstimmung zwischen einem

gescannten und einem gespeicherten Bild ermitteln lässt, wobei dieser Wert z.B. aussagt, ob ein gescanntes Bild überhaupt eine Person darstellt.

Der vorliegende Forschungsbericht beschreibt in den Kapiteln 2, 3, 4 und 5 die eingesetzten Verfahren im einzelnen und veranschaulicht an Beispielen die Wirkungsweise und die Eigenschaften der Algorithmen. Das Kapitel 6 stellt das Gesamtsystem vor und erläutert an statischen Testbildern den Verlauf einer Gesichtserkennung. Durch die graphische Anzeige von Teilergebnissen, Zwischenbildern und Fehlerwerten lässt sich der Ablauf der Erkennungsalgorithmen verfolgen und die Entscheidung, welches Gesicht das Computerprogramm am Ende erkennt, gut nachvollziehen.

Im letzten Kapitel arbeitet das Erkennungssystem im Echtzeitbetrieb. Die Kamera wird dabei automatisch nachgeführt. Zahlreiche Bildschirmausdrucke erläutern die Funktionsweise des Erkennungssystems im praktischen Einsatz.

Zu welchen beachtlichen Leistungen das entwickelte Gesichtserkennungssystem in der Lage ist, demonstrieren die in der folgenden Abb. 8.1 dargestellten Ergebnisse:

Abb.: 8.1: Gescanntes Bild (oben), erkanntes Bild (unten)

Die linke und die mittlere Darstellung zeigen Bildaufnahmen aus einer Sammlung von Testbildern, die im Graustufenformat vorlagen. Das rechte Bild zeigt eine Kameraaufnahme des Autors des vorliegenden Forschungsberichts beim Testen des Erkennungssystems. Die Vermummung half nicht, denn das Erkennungsystem hat die richtige Person erkannt.

Die hervorragenden Ergebnisse bei der Erkennung vermummter Gesichter entstanden unter günstigen und definierten Lichtverhältnissen innerhalb von Laborräumen. Daher besteht nun die Aufgabe, das Erkennungssystem für die rauhen Umgebungsbedingungen eines praktischen Einsatzes auf Fluren, Hallen und öffentlichen Plätzen weiter zu entwickeln.

Zur Entwicklung eines robusten Erkennungssystems eignet sich ein kleines ferngesteuertes Fahrzeug, das die Kamera trägt und die Bilddaten an einen leistungsfähigen Rechner überträgt. Für ein derartiges System bieten sich eine Fülle von Anwendungen an. Daher beschaffte der Fachbereich Informatik ein derartiges Fahrzeug, das unter der Bezeichnung „WiFiBot“ verkauft wird.

Das in der Abb. **??** dargestellte Fahrzeug „WiFiBot“ lässt sich über ein WLAN-Netzwerk steuern und ist 28 cm lang, 33 cm breit und 20 cm hoch. Die Bildscans gelangen zur Auswertung mit den Erkennungsalgorithmen in ein Notebook. Die Kamera ist an dem Aufdruck D-Link erkennbar. Die quaderförmigen Behälter nehmen die Akkus für ca. 90 Minuten Betriebsdauer auf. Die Elektronik ist in den schwarzen Radkästen untergebracht. Die vier Räder lassen sich unabhängig voneinander steuern und ermöglichen dadurch Bewegungen in alle Richtungen.

Abb.: 8.2: Das Fahrzeug „WiFiBot“ mit einfacher Kamera

Die Hardware besteht aus einem DK40 Ein-Chip Computer Modul mit 802.11b/g Standard, aus einem 10 BaseT Ethernet Switch mit fünf Ports, aus einem SC12 de Beck 186-20 Mhz AMD Prozessor mit 512 KBytes RAM, 512 kBytes Flash, RTOS sowie aus zwei seriellen Schnittstellen. Das Fahrzeug selbst ist mit zwei IR-Sensoren zur Abstandsmessung bestückt. In das RTOS integriert sind FTP, TELNET und WEB Server. Der FTP-Server dient zum Software-Upload. Möglich wäre auch der Betrieb mehrerer „WiFiBots“ über einen Access Point. Für die Kamera, den Ein-Chip-Computer und den Access Point stehen eigene IP-Adressen zur Verfügung.

Die Abb. **??** stellt in der linken Bildhälfte die Aufnahme einer Digitalkamera dar und man sieht, wie sich das „WiFiBot“-Fahrzeug einem Hindernis nähert. Die rechte Bildhälfte zeigt den entsprechenden Bildschirmausdruck. Die auf dem „WiFiBot“-Fahrzeug montierte und mit dem Fahrzeug gelieferte einfache Kamera lässt sich nicht verstellen, was die Anwendungsmöglichkeiten erheblich einschränkt. Daher ist eine Umrüstung auf eine um alle Achsen bewegliche Canon-Kamera geplant.

Abb.: 8.3: Die Steuerung des „WiFiBot“

Das Ziel der Forschungsarbeit besteht darin, ein mobiles System mit einer beweglichen Kamera zu entwickeln, das im Hinblick auf die Erkennung von Personen neben der möglichst hohen Erkennungsrate zusätzlich die für den praktischen Betrieb nötige Robustheit gegenüber wechselnden Lichtverhältnissen und Aufnahmeparametern aufweist.

Hannover, im Februar 2005

(Prof. Dr. Werner Lechner)

Kapitel 9

Anlagen

Abbildungsverzeichnis

Literatur: Gesichtserkennung

[1] Jain A., S. Li (2004): *Handbook of Face Recognition*, Springer Verlag, ISBN 0-387-40595-X.

[2] E. Dikich (2004): *Verfahren zur automatischen Gesichtserkennung* Logos Verlag, ISBN 3-8325-0428-1.

[3] Bennamoun M., Mamic G. (2002): *Object Recognition, Fundamentals and Case Studies* Springer Verlag, ISBN 1-85233-398-7

[4] Rakover S., Cahlon B. (2001): *Face Recognition: Cognitive and computational processes*, John Benjamins Publishing Company, ISBN 90-272-5151-7.

[5] Bartlet M.(2001): *Face Image Analysis by Unsupervised Learning*, Kluwer Academic Publisher, ISBN 0-79237-348-0.

[6] Jesorsky O., Kirchber K., Frischholz R. (2001): *Robust Face Detection using the Hausdorff Distance*, Third International Conference on Audio- and Video-based Biometric Person Authentication, Springer, Lectures Notes in Computer Science, LNCS-2091, pp. 90-95

[7] Lin K., Lam K., Siu (2001): *Locating the eye in human face images using fractal dimensions* IEEE Proceedings in Image Signal Processing, Vol. 148, December 2001

[8] Hallinan P., Gordon G., Yuille A., Giblin P., Mumford D. (1999): *Two- and Three-Dimensional Patterns of the Face*, AK Peters, Ltd., ISBN 1-56881-087-3.

[9] Sa M (2001) *Pattern Recognition - Concepts, Methods and Applications* Springer Verlag, ISBN 3-540-422-978

[10] Gotthalmseder M.(2003): *Visuelles Erkennen und Bildschaffen*, Uni Augsburg, www.bildschaffen.de

[11] Conde J. (2001): *Optische Raumüberwachung*, Disseration der Gesamthochschule Duisburg, Universitätsbibliothek, Campus Duisburg, Dokument duett-05252001-100856

[12] Fang Ch., Chen S., Fuh Ch. (2003): *Road-Sign Detection and Tracking* IEEE Transactions on Vehicular Technology, Vol 52, No. 5, pp. 1329-1341September 2003

[13] Milch S. (2001): *Videobasierte Fahreridentifikation in Kraftfahrzeugen*, Dissertation der TU Darmstadt

[14] Brünenberg K. (1999): *Untersuchungen zur Beleuchtungsabhängigkeit am Beispiel der Gesichtserkennung*, Internal Report 99-07, Ruhr-Universität Bochum,Institut für Neuroinformatik

Literatur: Neuronale Netze

[15] Spektrum der Wissenschaft, September 2004

[16] Samarasinghe S. (2004): *Neural Networks for Pattern Recognition in Scientific Data* Auerbach Publishers Inc., ISBN 0-849-33375-X

[17] Haykin S., Gwynn R. (2004): *Neural Networks: A Comprehensive Foundation* Verlag Prentice Hall, ISBN 0-131-47139-2

[18] Oldroyd M. (2004): *Opimisation of Massively Parallel Neural Networks* Fultus Cooperation, ISBN 1-596-822010-1

[19] Yegnanarayana B. (2004): *Artificial Neural Networks* Prentice Hall, ISBN 8-120-31253-8

[20] Bosque M. (2004): *Web-Based Neural Nets: Interactive Artificial Neural Networks for the Internet and Tricks* Verlag iUniverse.com, ISBN 0-595-31771-5

[21] Hsiao S. (2002): *Innovative Algorithms for Running a Wb-Based Pattern Recognition Search System for a Component Pattern Database*, Malaysian Journal of Computer Science, Vol. 15, no 2, pp. 78-93

[22] Wunsch D., Hasselmo M., Venayagamoorthy G., Wang D. (2003): *Advances in Neural Networks Research Ijcnn (2003)* Pergamon Press, ISBN 0-080-44320-6

[23] Gupta M., Jin L., Homma N. (2003): *Static and Dynamic Neural Networks*, IEEE Computer Society Press, ISBN 0-471-21948-7

[24] Reusch B. (2001): *Computational Intelligence, Theory and Applications*, International Conference, 7th Fuzzy Days, Dortmund, October 2001, Springer Verlag, ISBN 0-302-9743

[25] White J., Southgate E., Thomson J., Brenner S. (1986): *The Mind of a Worm*, Phil. Trans. Royal Soc. London, Series B, Biol Scien. Vol 314, Issue 1165, pp. 1-340, http://www.wormatlas.org

[26] Eigen M., Winkler R. () Serie Piper, ISBN 3-492-00710-4

[27] Hopfield J., Tank D. (1986): *Computing with Neural Circuits: AModel*, Science, vol. 233, pp. 625-633

[28] Amit D. (1989): *Modeling Brain Function: The World of Attractor Neural Networks*, Cambridge University Press, New York

Literatur: Bildverarbeitung

[29] Meyer-Bäse U. (2000), *Schnelle digitale Signalverarbeitung, Algorithmen, Architekturen, Anwendungen*, Springer Verlag, ISBN-3-540-67662-7

[30] Bäni W. (2002): *Wavelets* Oldenbourg Verlag, ISBN 3-486-25427-8

[31] Jähne B. (2002): *Digitale Bildverarbeitung* 5. Auflage, Springer-Verlag, ISBN 0-12345-123-5

[32] Sahbi H., Boujemaa N. (2002): *Robust Face Recognition using Dynamic Space Warping* Biometric Authentication, LNCS 2359, pp. 121.132, 2002 Springer Verlag, ISBN 3-540-43723-1

[33] Jensen A., Cour-Harbo A. (2001): *Ripples in Mathematics, The Discrete Wavelet Transform* Springer Verlag, ISBN 3-540-41662-5

[34] Wiskott L., Fellous J., Krüger N., Malsburg Ch. (1999): *Face Recognition by Elastic Bunch Graph Matching*, Intelligent Biometric Techniques in Fingerprint and Face Recogition, CRC Press, ISBN 0-8493-2055-0, Chapter 11, pp. 355-396

[35] Karpinski M., Rytter W. (1998): *Fast Parallel Algorithms for Graph Matching Problems* Oxford Science Publications, ISBN 0-19-850162-5

[36] Turk M., Pentland A. (1991), *Eigenfaces for recognition*, Journal of Cognitive Neuroscience, 3(1):71-86, 1991

[37] Stone J. (2004), *Independent Component Analysis: A Tutorial Introduction* Bradford Book, ISDN 0-262-69315-1

[38] Elagin E., Steffens J., Neven H. (1998): *Automatic Pose Estimation System for Human Face based on Bunch Graph Matching Technology*, Proceeedings of teh International Conference on Automatic Face and Gesture Recognition, pp. 136-141, Nara, Japan

[39] Swelden W. (1996): *The Lifting Scheme, A construction of biorthogonal wavelets*, Appl. Computer Harmon. Anal., vol. 3(2), pp. 186-200

[40] Loy G. (2002): *Fast Computation of the Gabor Wavelet Transform*, DICTA20002, Digital Image Computing Techniques and Applications, Melbourne, Australia

[41] Daubechies I., Swelden W. (1998): *Factoring Wavelet Transform into Lifting Steps*, J. Fourier Anal. Appl. 4, no. 3, pp. 245-267

[42] Mallat S. (1989): *A Theory for Multiresolution Signal Decomposition, The Wavelet Representation*, IEEE Trans. Pattern Analysis and Machine Intelligence,vol. 31, pp. 674-693

Literatur: Softwareentwicklung

[43] Eidenberger H., Divotkey R. (2004): *Medienverarbeitung in Java, Audio und Video mit Java Media Framework und Mobile Media API*, dpunkt.verlag, ISBN 3-89864-184-8

[44] Zeppenfeld K. (2004): *Lehrbuch der Graphikprogrammierung* Spektrum Akademischer Verlag, ISBN 3-8274-1028-2

[45] Dunkel J., Holitschke A. (2000): *Softwarearchitektur für die Praxis*, Springer Verlag, ISBN 3-540-00221-9

[46] Bengel G. (2003): *Verteilte Systeme, Client-Server-Computing für Studenten und Praktiker*, Vieweg Verlag, ISBN 3-528-15738-0

[47] Jesse R. (2002): *Java-Swing*, Verlag Moderne Industrie Buch (Cbhv), ISBN 3-8266-7224-0

[48] Krüger G. 2002): *Handbuch der Java-Programmierung*, 3. Auflage, Addison-Wesley Verlag,ISBN 3-8273-1949-8

[49] Oechsle R. (2001): *Parallele Programmierung mit Java Threads*, Fachbuchverlag Leipzig, ISBN 3-446-21780-0

[50] Chandra R., Dagum L., Kohr D., Maydan D., McDonald J., Menon R. (2001): *Parallel Programming in OpenMP*, Morgan Kaufmann Publishers, ISBN 1-55860-671-8

[51] Willms R. (2000): *Java Programmierung Praxisbuch*, 2. Auflage, Franzis Verlag, ISBN 3-7723-7606-1

[52] Mittendorf S., Singer R., Heid J. (2003): *Java Programmier Handbuch und Referenz*, 3. Auflage,dpunkt.verlag, ISBN 3-89864-157-0

Index